



Universiteit Leiden

Opleiding Informatica

GPS referencing and Data Storage Manipulation

Name: Jelle Sinnige and Tim van Polen

Date: 23/08/2019

1st supervisor: Fons Verbeek

2nd supervisor: Irene Martorelli and Leon Helwerda

BACHELOR THESIS

Leiden Institute of Advanced Computer Science
(LIACS)

Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

A common problem when working with digitized transcribed location data is that these databases are often incoherent and incomplete. Another situation which can result in incoherent and incomplete databases, is when multiple databases are merged together and used as one. Both these situations result in a lot of hassle to transform them by hand and make the final result clear and coherent.

During this research project we have worked on creating a system to transform these data-inputs into coherent and complete databases as well as find a method for the checking and improving of GPS location data. The designed system is able to perform these operations together as well as separately and creates an easy and effortless process for the user. The system works efficiently as well as accurately and is designed to be sustainable and expandable, by making use of several design implementations.

Contents

Abstract	2
Contents	3
1 Introduction	5
1.1 GPS Referencing and Data Storage Manipulation	5
1.2 Research questions	6
1.3 Thesis overview	7
1.4 Work division	7
2 Material and Method	8
2.1 JavaScript	8
2.2 JSON	8
2.3 Node.js	9
2.4 Google Geocoding API	10
2.5 Database types.....	11
2.6 GPS coordinates	12
3 Design	14
3.1 Design choices.....	14
3.2 System overview and major components	20
4 Implementation	24
4.1 Terminal interface.....	24
4.2 Command and Data flows.....	25
4.3 User Instructions	29
4.4 Main modules	30
4.5 Support modules.....	34
5 Experiments	40
5.1 Support modules validation tests	40
5.2 Location module tests.....	45
5.3 Database module tests	52
6 Conclusion, Discussion and Future Implementations	56
6.1 Conclusion.....	56
6.2 Discussion.....	58
6.3 Future implementations	60
Bibliography.....	61
Appendix A – Readme	63

Appendix B – Configuration Syntax Guide	67
Appendix C – Usage example	74
Appendix D - Support modules validation tests	93
Appendix E - Location module tests	100
Appendix F - Database module tests	107

1 Introduction

This chapter will provide a short introduction to the topic of the thesis and define the corresponding research questions and sub-questions. Furthermore, we will give an overview of the remaining chapters of the thesis and discuss how the work has been divided.

1.1 GPS Referencing and Data Storage Manipulation

Over time a lot of researchers have gathered a lot of information about different organisms on our planet. This type of data is mostly stored in databases with manually transcribed records. A lot of this stored (historical) information is used for further research. To make use of these databases for further research it is of utmost importance that the information within the databases is not only complete but also coherent.

The project regarding our thesis will focus on two aspects regarding the process in making these databases complete and coherent as well as checking the information included for errors.

The first problem we will tackle in our thesis is the fact that a lot of these, possible historical, databases contain location data points. This location data can be incomplete and inconsistent. Furthermore, different databases can use different formats to store these coordinates. Our thesis will focus on this problem and provide a way to transform these location data entries to a uniform system so these can be easily combined. Also, a checking system will be implemented to easily filter out and alter the incorrect data entries. The to be designed system will contribute towards increasing the consistency and coherency of the databases.

Another frequently occurring issue when dealing with databases is altering and checking the containing data and combining information of multiple databases together. Multiple options arise to tackle this process. It can be done manually, or by writing extensive queries to handle the problem. This process can be tedious, time consuming and is still very prone to human error. Especially when dealing with databases with a large number of entries problems can arise easily. The second process of our application will provide a way to easily combine databases together and manipulate the data in an orderly and easily managed fashion. Furthermore, it will make it possible to check for errors and duplicates. Lastly the system will be compatible with different types of databases and will also be able to transform the data to another type when needed.

1.2 Research questions

The research questions we will try to answer at the end of this paper are stated below. To improve our ability of providing a complete answer to the 3 research questions, we split them up into sub-questions where required.

1.2.1 Research question 1

“How can we find a method to correct location data in existing databases and provide an easy process for further data entries?”

1.2.1.1 Research sub-question 1.1

“What is the fastest and most reliable way to add and alter the location data in the existing database, making this data complete and consistent?”

GPS coordinates can be stored using a number of different format styles, or this information can be non-existent in certain data entries. We will need to find a method to create a complete and coherent output from these possibly inconsistent input files.

1.2.1.2 Research sub-question 1.2

“What formatting and input guidelines should be used for future data entries to keep the location data of the entries added to the database complete and consistent?”

Not every exception can be accounted for within the system, therefore some restrictions and guidelines will have to be added towards the formatting of the input.

1.2.2 Research question 2

How can we provide a process to easily organize, transform, combine and check databases?

We need to find a method to best transform, combine and check databases combined or separately. The method needs to work in such a way that processing large files will run smoothly and consistent.

1.2.3 Research question 3

“How can we build a sustainable system that can add and verify GPS location data to databases while also being able to organize the data, and is compatible with different kind of databases?”

1.2.3.1 Research sub-question 3.1

“How can we design the system to operate both processes effortlessly alongside each other within one system?”

The system will need to perform two different processes, which will need to be able to work together as well as individually.

1.2.3.2 Research sub-question 3.2

“How can we design the system to be sustainable?”

The system needs to be designed in such a way that it can easily be expanded with extra functionalities as well as being reliable, consistent and can be used by multiple (types of) databases.

1.3 Thesis overview

In the following chapters we will first explain thoroughly which materials and methods we have used while creating the application. In chapter 3 we will discuss the design of the system and its core processes, as well as substantiate on the made design choices. In chapter 4 we will provide you with the actual implementation of the clarified design of chapter 3, as well as some additional information to be able to use the system. In chapter 5 we will substantiate on our final implementation with an overview of the testing process and the outcome of the different sets of tests. Followed by the conclusion, discussion and future work section. At the end of the paper you will find the bibliography followed by the appendixes.

Chapters 2 and 3 mainly focus on the design aspect of the research questions, where chapter 4 substantiates further on this on a more technical level. Chapter 5 is used to elaborate on the substantiation on the claims made in the previous chapters, and help answering the research questions.

1.4 Work division

During this project we have worked in a team of two. We divided the workload into two and both worked on our parts of the project. Tim mainly worked on the GPS location transformation system and Jelle mainly worked on the database transformation part.

The system contains other support and overarching modules which were created and tested in cooperation. Though Jelle did focus more on the development and Tim on the testing part of the project.

However, most work was performed together, we assisted each other intensively during the project. As a result, we created both the application and thesis in cooperation.

2 Material and Method

In the following chapter we will discuss the materials, API's and libraries used for the development of our application. Per used material we will first discuss what it does and then what we used it for in our application, followed by one or more alternatives.

2.1 JavaScript

JavaScript is a widely used interpreted programming language, which means that most of its implementations execute instructions directly, without the need to first compile the language into machine-language instructions before the program can be run. Therefore, this language is one of the core technologies used in the World Wide Web, alongside HTML and CSS. The language is mostly used to make webpages interactive or to create web applications. (Javascript, sd)

There are a couple reasons that made JavaScript the programming language we have used for our application.

- One of the key aspects of our application is that it can be used in varying circumstances. It was very important that the application could be used on every computer and web browser. This aspect was most easily achieved using the JavaScript programming language.
- The great number of available libraries within the JavaScript programming language also contributed towards our decision. This wide variety of libraries available to incorporate in our application, made the development easier and more focused on the crucial parts. Node JS, one of the libraries used, will be discussed in more detail in subsection 2.3.
- The fact that JavaScript does not work chronically but event based (Javascript, sd). Allowing to run multiple threads at the same time, reducing our computing time.

The main alternatives to JavaScript would be to use a compiled programming language, such as C++. However, the problem with using a compiled programming language would be that the whole program should be initialized first before use. The use of interpreted language gave us more flexibility within the application.

2.2 JSON

JavaScript Object Notation is a standardized data-interchange format (Introducing JSON, sd). It is easy for people to read and write information in JSON and easy for machines or applications to parse and generate. JSON is a text format that is not language dependent (Introducing JSON, sd), which make it ideal for data exchange between different programs and or languages. It is built on two structures; a collection of pairs of names and values and an ordered list of values.

We use JSON to store all the information coming into and within the application. The fact that JSON is language independent, makes it a good fit to use as storage system within our application. It makes interaction and implementation of and with other applications very easy and thus makes our application more widely applicable.

An alternative would be to set up a MySQL database. However, all the attributes of our application would then have to run on our own database. With using JSON format, we can easily use external data to run the visualization part of the application, as well as communicate between the different parts of our application.

2.3 Node.js

Node.js is a software platform on which applications can be developed and run on (About Node.js, sd). Those applications have to be written in JavaScript. Node.js is an asynchronous event driven JavaScript runtime and designed to support scalable network applications (About Node.js, sd). However, in contrast to most JavaScript applications they do not have to be executed in a web browser but are executed in the JavaScript-Engine of Node.js (About Node.js, sd). The applications can run on any PC with Node.js runtime installed. Also, Node.js offers an alternative method of server-side scripting with its built-in HTTP-server, which makes it possible to run a webserver without the use of Apache or Light.

With the use of Node.js we were able to create a webserver without the use of any other languages besides JavaScript. Furthermore, because Node.js is asynchronous event driven. This gives us the opportunity to run different processes simultaneously, increasing the efficiency of the system. We also used a number of Node.js modules, an overview of the most important modules used will be given below.

2.3.1 Node.js File System Module

The Node.js File System Module allows the system to read, create, update, delete and rename files in the system (Node.js File System Module, sd). This allows us to create a database like system based upon JSON files, to keep track of the data available in the system.

2.3.2 Node.js HTTP Module

The built-in HTTP Module of Node.js allows us to transfer data over the Hyper Text Transfer Protocol (Node.js, sd). In this way we are able to create a webserver, to run all the applications on. In this way the applications can be accessible at all times and from all computers and web browsers.

2.3.3 Node.js HTTPS Module

The Node.js HTTPS Module allows us to transfer data. Instead of the HTTP module this type of data transfer will be secured, using a TSL/SSL protocol (Node.js, sd). We will preferably set up this type of connections when setting up the database, to create a secure and safe webserver.

2.3.4 Node.js Query String Module

The Node.js Query String Module allows us to parse post data (Node.js, sd).

2.3.5 Node.js Request Module

The Node.js Request Module can best be described as a wrapper around Node.js built-in HTTP module (Node.js, sd). All the functionalities it provides, can also be achieved with the use of just the HTTP module. However, the Request Module makes this whole process a lot easier.

2.3.6 Node.js Child Process Module

The Node.js Child Process Module allows us to create different child processes (Node.js, sd). Which allows us to run multiple processes within one application or run various programs on the server. This ensures that if one part of the application has errors or problems, only that subprocess is affected by these problems. The rest of the application can continue running.

2.3.7 Node.js Net Module

The Node.js Net module supports inter-process communication (IPC) by creating a network API for creating stream-based Transmission Control Protocols or IPC servers (Node.js, sd). In our application we will use this module to control the ports.

2.3.8 Node.js CSV-parse

The Node.js CSV-parse module will make it possible to covert CSV text into objects and arrays (csv.js.org, sd). This module will make it possible for us to transform incoming CSV data into other types of database structures.

2.3.9 Node.js CSV-stringify

The Node.js CSV-stringify module is the counterpart of the Node.js CSV-parse module (csv.js.org, sd). It makes it possible to convert data entries, arrays and objects into CSV text. This module will be used to transform data handled by our system into CSV output text.

2.3.10 Node.js MySQL

The Node.js MySQL module makes it possible to connect with a MySQL type of database via a Node.js code/connection (NPM, mysql, 2019).

2.3.11 Node.js MonetDB

The Node.js MonetDB module makes it possible to connect with a MonetDB type of database (NPM, monetdb, 2016).

2.3.12 Node.js MariaDB

The Node.js MariaDB module makes it possible to connect with a MariaDB type of database (NPM, mariadb, 2019).

2.4 Google Geocoding API

The Google Geocoding API is an API to transform location information into geographic coordinates. It is also possible to use the API to perform this process in reverse and receive location information based on geographic coordinates (Google, Developer Guide, sd).

Within the application the Geocoding API will be used in checking user provided coordinates based on the included location information as well as providing coordinates when they lack completely in the provided information.

Because of this main purpose the Google Geocoding API was chosen to be implemented into the application. When providing the API with areas of interest, the API returns a set of geographic coordinates based on the edges of the provided area. These coordinate sets can then be used to check if a user provided coordinate is within the specified area.

An alternative of the Google Geocoding API would be to make use of the Geonames API for the purpose as explained previously. A problem with making use of the Geonames API is that instead of returning a set of coordinates for the marking of an area, the API only returns a single point at the center of this area (Web Services, sd). This would require the application to have knowledge about the size and shape of the geographical body of interest, to be able to determine if the coordinates fall within the expected area. The Google Geocoding API requires less knowledge by creating an area of interest. Therefore, the use of the Google Geocoding API reduces the complexity of the algorithm.

2.5 Database types

To make the system as universally applicable we have chosen a number of wide applied database system types. Also, we have chosen to implement a couple of text-based storage applications as those are both widely used for storage and transition of databases. In the following subsections the implemented storage systems will be discussed and explained. Current implemented database systems are directed at database types used for storing datasets for which the application was designed initially. Our focus on implementing multiple databases is directed at making the software as usable and generic as possible, so a wide variety of systems can make use of it.

2.5.1 MonetDB

MonetDB is an open-source column-oriented database system, and was developed in the Netherlands (MonetDB.org, sd). Its design is implemented in such a way that it delivers high performance when operating on complex and extensive queries containing a very high amount of data entries. Within Leiden University this system is used for research and storage of large databases containing lots of different kind of information. Further use of the system will be most probable within Leiden University, we have chosen to make sure the system is compatible with MonetDB.

2.5.2 MariaDB

MariaDB is one of the world's most highly used database systems, coming in at a twelfth place worldwide in popularity (DB-Engines, 2019). MariaDB is called a relational database management system (RDBMS). This type of database management system saves their records in tables, with rows containing the records and in the columns the information which has to be saved per record (MariaDB, sd). MariaDB is also mostly compatible with programs using the MySQL system. We have implemented this system as one of the databases provided for testing the system will be making use of this type of database management system, furthermore as mentioned previously, it is widely used worldwide and therefore increases the chance of compatibility of our system with the to be used databases.

2.5.3 MySQL

MySQL is, as mentioned in the previous section, alike MariaDB. MySQL, however, is even more popular worldwide than MariaDB, coming at a second place in the worldwide popularity ranking of databases management systems (DB-Engines, 2019). MySQL is also an opensource RDBMS with at its core the SQL language to build up databases and performing queries (MySQL-Enterprise, sd). At first MySQL was mostly used by internet applications, in most cases in combination with the programming language PHP. Nowadays it is used in many other applications. We have chosen to implement this system in our application because of compatibility with other database system types.

The following two implementations are not database systems but are widely used to store data and tables. Therefore, we have chosen to make our system also compatible with these two types of storage options.

2.5.4 CSV

The comma separated values file type is a widely used implementation to store tabular data (examples of this are spreadsheets and databases) (Kommagescheiden bestand, sd). As the name suggests it is a filetype that stores the records in rows, with the information per record separated by a comma or other type of delimiter. The CSV filetype is also compatible with Microsoft Excel and other spreadsheet-applications. The main reason for us to allow implementation of this file type is that a number of the provided test-databases were provided in this format. Furthermore, this type of data storage is widely used and known worldwide making it a great asset to be compatible with our system.

2.5.5 JSON

Lastly, we will discuss the JavaScript Object Notation (JSON). As mentioned in section 2.2, JSON is at the core of our system for exchanging data between the different operating procedures (Introducing JSON, sd). It allows for an easy and structured data transfer process between the webserver and web applications. Furthermore, it is a language independent data format, although initially designed for JavaScript. As this will be the core language used for the design of our application, incorporating JSON also as an input/output option is of a great essence and easily implemented into the system.

2.5.6 Future implementations

Although currently the system is designed to only connect, receive input and deliver output of data in the beforementioned database system types. The system will be designed in such a way that any other database system type could be added to the system. One of the methods this will be provided, is by making use of the available Node.js modules for the connection with databases. Including additional modules to increase the number of database systems the application can operate with will therefore be possible.

2.6 GPS coordinates

GPS coordinates consist of two numbers, the latitude and longitude. Together these two create a point on a map (Geographic coordinate system, sd). If this data is already available, our application checks the GPS coordinates, if they correspond with the country, area or city given in the received data. The application will perform this operation by creating a bounding box around the area, based on the given geographic coordinates by the Google Geocoding API. If the coordinate of the user is outside of this box the coordinates will be improved to the center of the area. If no coordinates are provided by the user, this information will be added by the application by again providing the coordinates of the center of the created box.

Two main types of storing GPS coordinates exist. The first one, called decimal degrees (DD) expresses latitude and longitude coordinates as (either positive or negative) decimal values and are very common in a wide array of geographic information systems (Decimal degrees, sd). This type of GPS coordinates expressing is also used in a lot of mapping type of web applications, among which is the Google Geocoding API. Another way of storing GPS coordinates is making use of degrees, minutes and seconds (DMS) type of expression for storing GPS coordinates (Geographic coordinate system, sd). Within our system we will make use of the DD type of expression for latitude and longitude

coordinates, because of the uniformity and ease of use. However, we will make the system compatible with input of both types of latitude and longitude expressions interchangeably. The system will translate all of the input geographic coordinates into the DD type of notation.

A great disadvantage of the DMS expression system over the DD expression system is the variation of notation possibilities within the expression system, making it hard to determine how the given information should be interpreted (DMS vs DD, 2018). Possible misinterpretations that can happen:

- North/south and east/west can both come as the first coordinate
- N/S and E/W can both be replaced by a plus or minus sign as the notation
- Sometimes one of the 3 values in the notation that equals to zero is omitted

Therefore, we have chosen to make our system work with the DD type of expression system, although the system will be able to use input of the DMS system type of expression.

3 Design

In the following chapter we will discuss the design and implementation of the system divided over 2 sections. In the first section we will couple the research questions and sub questions, together with the materials and methods used in an overview in which the major design choices made, will be discussed and explained. Following we will further clarify the system by giving an overview of the total system and its major components and processes.

3.1 Design choices

In this section we will elaborate on which choices were made when designing the system and what made us choose for certain implementations. We have divided this section into a number of subsections. In each subsection we will discuss one of the choices made when designing the system or in what way we made sure certain aspects were to be reckoned with.

3.1.1 JavaScript / Node.js

JavaScript is the language we will use when designing our application. As previously discussed in chapter 2 JavaScript is an ideal language when creating a web application.

There are, however, a couple of drawbacks to using JavaScript over other types of programming languages:

- First of all, JavaScript is relatively slow compared to a compiling programming language. When running a compiled language, all compilations are done prior to running the program therefore these are mostly much faster at executions. The problem however is that C++ or other languages of a similar type, do not react well with asynchronous web applications. Which is a part of the design of our application.
- Secondly another drawback is that flexible typing can cause unexpected results and behavior within the program. Sometimes problems can occur with timing certain processes, as they will need information provided by a previous subprocess which is not finished yet. This can cause for unexpected results and behavior. We are aware of these problems and counteract them we implemented buffer times when possible.

As previously mentioned, an alternative programming language for the application would be c++. However, when using c++, we would have to take into account the operating dependent libraries and the system would be less flexible and sustainable. Another alternative would have been Java. However, Java is not built for asynchronous web applications as well as having a less extensive libraries as JavaScript, therefore JavaScript was the obvious choice for the design of this application.

Another software platform widely used within the system will be Node.js. This software platform will allow us to enhance the use of JavaScript with applications we normally would have to use different languages for, to create the same effect. The wide library of plugins and modules of the Node.js system allows us to incorporate all these functionalities within our web application with only the use of the JavaScript language.

3.1.2 JSON

Within the application data is stored and send within and between modules by making use of the JSON format. We have chosen for this type of data storage and transfer format, because it is specifically designed for the JavaScript programming language. Furthermore, it is very easy to read and write information in the JSON format, not only for users but also for generation programs. Once the structure of the JSON files used within the system is clear, it can be very easy to create an algorithm to generate these files for the use of the system. This will only enhance the efficiency of the system further.

3.1.3 Modularity

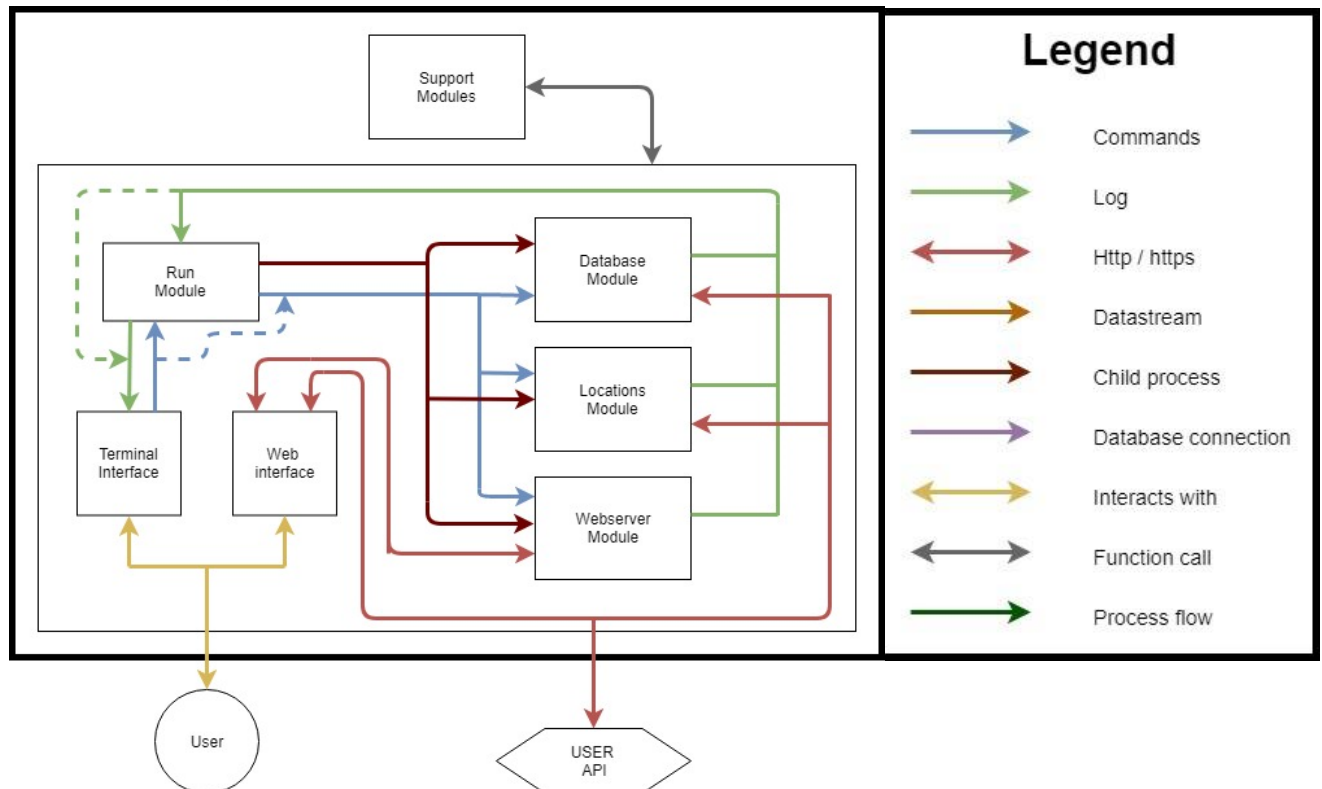


Figure 1 - System overview

The following design choice becomes apparent when looking at figure 1 and will also be further in detail explained in subsection 3.2 and chapter 4. The system and the separate modules will be designed in such a way that they are able to work/ and cooperate together when using the complete system at once. However, it will also be possible that for a certain application only one of the modules needs to be used. Therefore, the different modules will also be able to be used separately. This design choice will make the system applicable for more applications and will run more efficiently when used for targeted applications.

3.1.4 Types of connections

The types of connections to communicate within and with the system are stated in the legend of figure 1 and will be more in depth discussed in chapter 4. We have chosen to make the inter-process communication as universal as possible, in this way the information and signals needed per program are very similar across modules. Furthermore, in this way it is easier for the User and other programs to make use of the application. We will incorporate an API connection to make sure external programs can easily connect and make use of the system. Lastly by designing it in this way, it will be easier for future implemented modules to interact with the system.

Other connection types are a database connection made when connecting a database to the system. In this way the system will be able to directly perform its transformations into the connected database system. We have implemented this in such a way that it will be able to connect with different types of databases and that it will be possible to easily add other database system types as connections. There will also be a data file In and output connection which will make it possible for an external datafile to be altered by one or both of the main modules.

3.1.5 Child processes

As can be seen in figure 2 and 3 and also explained in subsection 3.2, we have chosen for the 2 major modules to incorporate child processes for the actual applications of the relevant modules. There are a couple of reasons we have chosen for this type of set up for the design of these modules. First of all, this allows for the system to run more processes of the same type at once, increasing the speed and efficiency of the application. By running the same subprocesses in parallel we allow for the same task to be processed faster and thus contribute towards the efficiency and sustainability of the system as a whole. Another reason for choosing to incorporate child processes in this way, is to keep the parent process free and accessible. Especially when processing large and multiple batches of data at once, in this way the system can capture all the information at once and start with the translation processes at the same time for all the batches. If we would try to process the same information without the use of child processes, it would take a lot more time to process all the information all the while the relevant module will be unavailable for other requests. Therefore, the application of child processes does not only increase the efficiency of the application but also makes the module available for further entries while processing simultaneously.

3.1.6 GPS location data control

The locations module will be able to add and check GPS location data. The module will make use of the Google Maps API when checking and adding the GPS data to the existing data in the datafile. When checking for the GPS data, the Google Maps API returns a single set of GPS coordinates when looking for a small area or point on a map. It will return two sets of GPS coordinates when looking for a bigger area on a map, which will contain a set of the most outlying points on the map, creating a rectangle area containing the requested country/area/city.

After this the application will check if the provided geographic coordinates correspond with the returned area based on the provided information. If these correspond, the old coordinates will be stored in the output. If the provided coordinates do not fall within the created bounding box, the coordinates are updated to the center of the area.

The application will provide the option to make the provided set of location information binding, or to give the option to increase the search area if possible. Providing the user the option to see either at which extend a coordinate corresponds with the given information or improves the given coordinates as soon as they mismatch with the drawn area. The system will make use of a user-determined margin of error, which can be altered depending on the type of data used.

There are a number of complications we will have to face when designing this part of the application. The first one is when creating the area for the control of the GPS data, a square is created of the outmost points. However, this is not completely accurate when controlling GPS data. For example, when looking at countries, this area can include neighboring countries or parts of the sea the country is adjacent to. We tried to counter this problem with the design of working bottom-up, however this part is not fool-proof and will be further discussed in the discussion.

Another complication is the fact that when working with the Google Maps API only a certain number of requests (up to 200 euros) is free per month which equals to around 40000 geolocation requests

per month. In most cases this should be enough, but when working with large databases it could be the case this number of requests is being transcended. In this case the application cannot further check/add GPS data to the datafile for that time period. This problem can be omitted by coupling a bought API key to the system.

3.1.7 Database direct insert

When controlling and transforming database record information we will be implementing a database direct insert type of system. When using this type of database insertion method, the system will constantly be in direct contact with the linked database system. In this way we will be able to directly check for records and insert records into the database system, without storing them in the cache of our own system.

The main advantages of this type of database system type are:

- We will not have any overlapping ID's between databases or the database and our own cache. In this way it will be less prone to making errors when altering the database while making use of the system for transformations. This will increase the overall sustainability and security of the system.
- The second advantage is that in case the system crashes, the already handled data will be stored in the coupled database system and will be saved. This will reduce the amount of information needed to be done when restarting the system and continuing where the system crashed.

A possible alternative to the direct insert database insertion method is the cache first insertion method, which has a couple of advantages and drawbacks compared to the direct insert method.

Advantages:

- Is faster when controlling and manipulating data. As it stores everything in the cache of its own system, this method loads and manipulates data faster than the direct insert method.
- Uploading bulk quantities of data will be faster when done at once instead of by using multiple individual connections.

Disadvantages:

- Lose all data in case of a crash. The system would then need to redo the entire operation.
- Does not take into account that data can change in the to be connected database during the processing of the system, which can result in duplicate information. This can cause errors in the database, which would be a major inconvenience.

As can be seen there are both some great advantages as well as disadvantages to using the cache first method. However, we have chosen for our system to be more reliable rather than to increase the processing speed. When working with large databases, first and foremost must every record be processed correctly and as reliable as possible.

3.1.8 User configuration files

When performing multiple data transformations on possible multiple databases all at once, a lot of different parameters are needed to make the process run smoothly. By implementing a certain standard protocol for the processing of these transformations, we will create a situation where this will be possible and as efficiently as possible. Furthermore, because we will be making use of the JSON type of notation, it will be very easy to create a simple generation program for the creation of the configuration files for the respective modules. Overall the configuration files will increase the usability and efficiency of the system as a whole.

3.1.9 UI en terminal interface

When designing the system, we will first focus on the working parts of the system. The location transformation module and database transformation module will be the priority of the system. Therefore, initially we will design the system to be working with a terminal interface, which allows the user to give commands via a terminal interface which the system will handle and will act accordingly. In chapter 4 these commands will be discussed more in depth. We will also try to create a user interface environment to increase the ease of use of the system, although this will not be a priority of this project.

3.1.10 Error and crash handling

The system will be designed in such a way that most of the errors and crashes will be omitted. However, when one of both will occur, the system will need a method to deal with these types of situations. The error handling will mostly consist of a clear signal connection with the terminal interface. Within this environment the user will be able to see every type of error and can act accordingly. Also, when working with datafiles, a separate datafile will be created consisting of the inputted data which the system was not be able to handle.

The modular design of the system will make it possible for the system to handle crashes in a sustainable and efficient way. Because of the design, it will be possible that if one of the modules or sub-modules crashes, only that part of the application will malfunction and the rest can still continue to work. It will then be further handled as an error within the system and send signals to the terminal interface to alert the user.

3.1.11 sustainability

When designing the system, it was very important to design it in such a way that it was sustainable and easy to be used with as much database system types as possible. Sustainability means the system should be compatible for use of current technology and database system types as well as easily be made compatible for any future updates and technologies. We will try to achieve this by a couple of design implementations, of which a number are already mentioned in the previous subsections. Following we will provide an overview of the (most important) design implementations regarding sustainability:

- The first implementation is the modularity of the system. The major components can be used separately or the system can work together as a whole. In this way the system is compatible for as many applications as possible. Another advantage of this type of design is that the inter-process communication needs to be standardized, therefore it is easier to implement further modules and applications for the system in the future.
- A second design implementation made towards increasing the sustainability of the system is the separation of the support modules from the major modules. These modules contain processes used by several of the major components of the system. By separating these

processes, they can be used in all the components by simple function calls. They will also be easier to be used for further future design implementations.

- When designing the database module, we designed it in such a way that it would be able to connect with as many database-system-types as possible. By making use of node.js implementing any future database system type can be done by making use of one of widely available modules provided for the node.js system. Therefore, it will be possible to use the system, with some alterations, with almost any database system type available.
- Lastly the design to let the two major components make use of child processes contributes towards the sustainability, as also already mentioned in 3.1.5. By keeping the parent process available for handling the requests from the user or API and creating one or multiple child processes per request, it keeps itself available for further requests at all time. In this way, the process will be able to handle much more requests at the same time and work much more efficient.

3.1.12 Efficiency

To increase the efficiency of the system, a couple of design implementations will be made. These implementations will have an impact on the overall performance of the application.

- The first implementation is that of the above-mentioned child processes. By implementing this type of processing, we will be able to perform multiple processes, transformations and calculations at the same time using multiple cores at once. This will increase the speed of the overall process.
- Another implementation to increase the efficiency of the system is that of the pipeline that will be created in the location helper process. As can be seen in figure 3, there exists a constant interaction between the file processor unit and that of the location check and correct unit. We will implement a pipelined structure to increase the efficiency of this interaction. This will increase the speed of the overall process of the location helper and location modules.
- Lastly, we will make use of user configuration files to streamline the process beforehand. This will make it possible to increase the speed especially when processing large amounts of information. By initializing all the transformations to be done beforehand in both the database and locations module, the system can then as efficiently as possible process all the data in a reliable and fast way.

3.2 System overview and major components

In the following section we will discuss the major processes, components and data flows within the designed application. We will first discuss a rough overview of the overall system, followed by a more in-depth overview within the locations, database and webserver modules.

3.2.1 System overview

The system is designed in three major components. First of all, run.js is the center of the program. In most cases, the user will connect with the system through this component to make further use of the different modules. Run.js establishes a connection between the user and the system via a terminal interface, handling the commands given in the command prompt. Also, a web connection is created between the user and the locations and database modules via the web interface. The web interface is hosted via the webserver module. It will also be possible for an external program or web application to connect with the system via an API connection as can be seen in figure 1. Furthermore, a user will be able to connect directly to one of the modules via the terminal interface, if desirable.

When starting the program, Run.js initiates all the processes and constantly interacts with the main modules via two types of connections.

1. **Command channel** will be transferred from the user to the concerning module via the run.js module.
2. **Log/signal channel** will be transferred back from the modules towards run.js to be send back to the user. A further explanation of the logs and signals can be found in section 4.

Run.js besides translating instructions from the user for the modules also handles the start-up and crash handling of the system. In this way the system will not crash completely in case of crashes or errors in one of the main modules or their child processes.

3.2.2 locations module

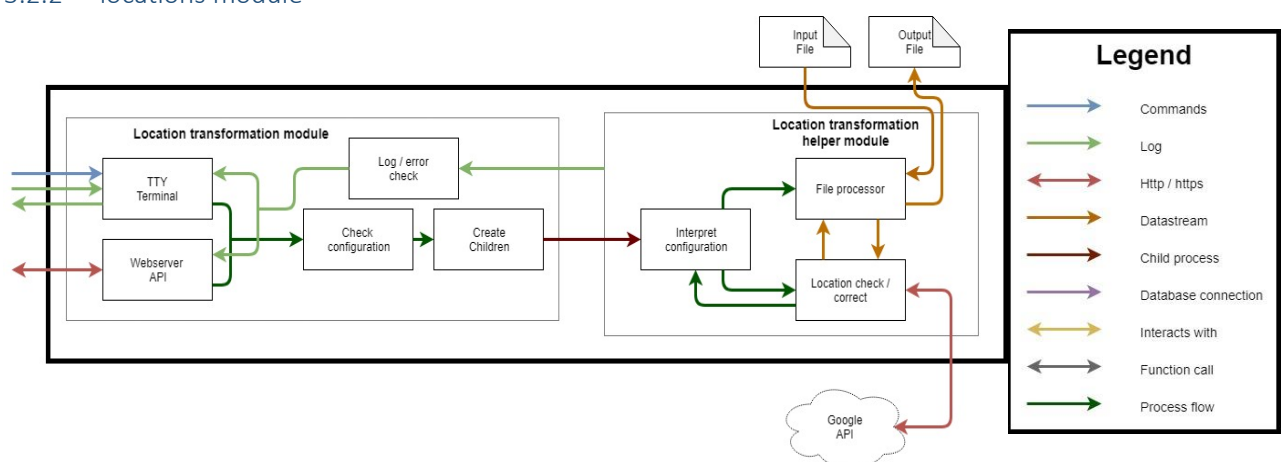


Figure 2 - Location module overview

The locations module handles the process of transformation and validation of the coordinates of a connected database file, an overview can be seen in figure 2. The user and/or the run.js module will connect with the module through a terminal interface and a webserver connection. Via these

connections the user will be able to send commands towards the module and connect or insert their data with/into the system. For every file that has to be processed, the create child function will create a child-process which will perform the actual check and transformation process. All the settings needed for the module to work in a proper way need to be specified in a configuration file (which will be discussed in chapter 4). Within the location-helper (the child process) every file that needs to be processed will be handled by its own location-helper process. This setup ensures the greatest efficiency and keeps its parent process open for further requests.

Within the location helper process the file processor unit will transform the input file in such a way that the location check / correct unit can process the information correctly. The location check / correct unit then creates an http connection with google to perform the actual check of the data and sends the received data back to the file processor unit. The connection between the file processor unit and location check / correct unit is a pipelined connection to increase the speed and efficiency of the process. Once the data is checked and again processed by the file processor unit it is then send to an external output file to be received by the user or API. Also, if in any case a part of the data did not process correctly, these will be outputted to another external datafile with a summary of the data containing errors. The DMS/DD transformations and check will also be done by the location check / correct unit.

During and at the end of the child processes signals and log is being send back to the TTY / terminal interface of its parent process. Which will then communicate this information back to the run.js module or directly back to the user.

3.2.3 Database module

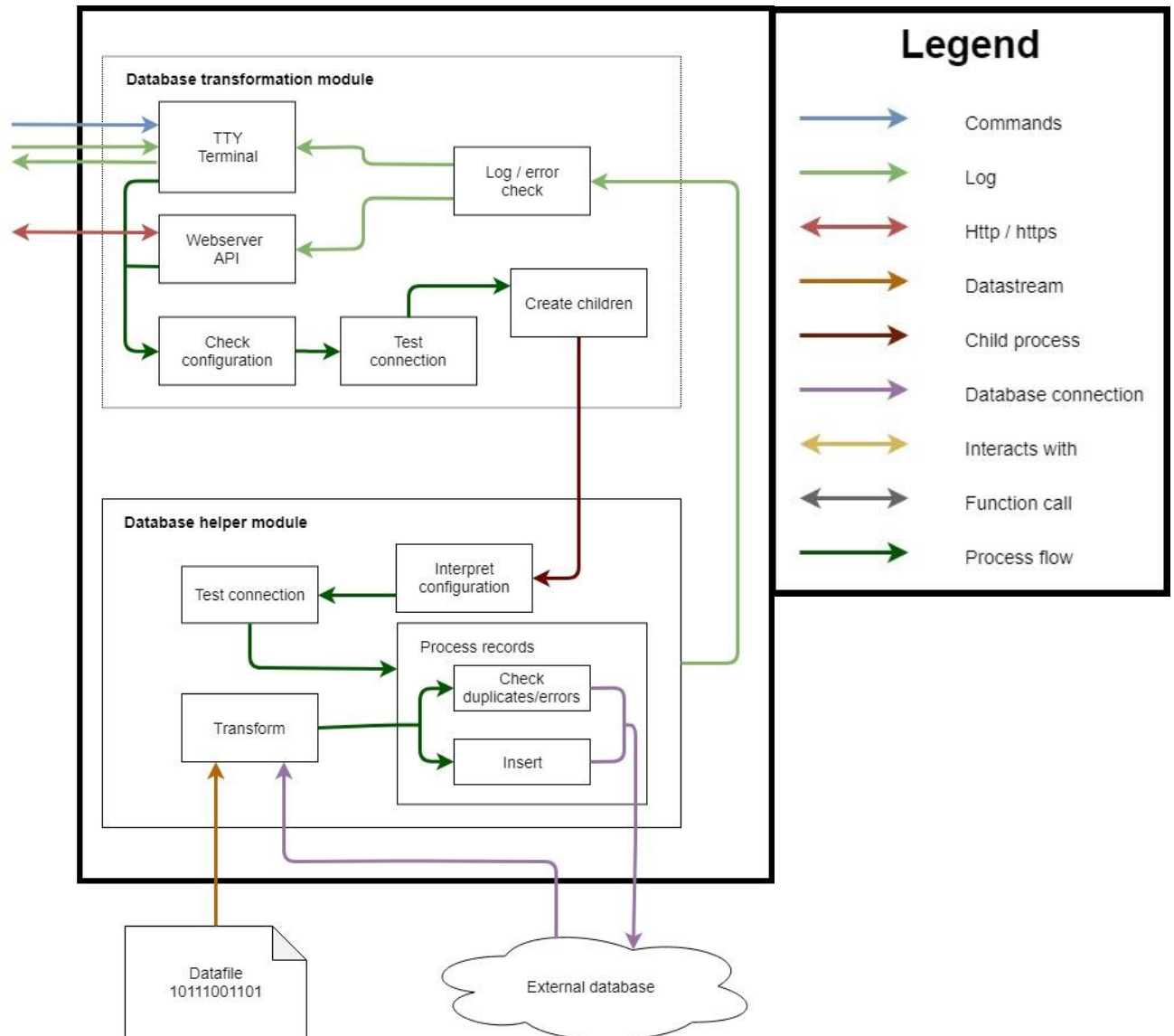


Figure 3 - Database module overview

The database module will handle any type of action regarding the adjustment of databases or database files. It is possible to merge and use different tables and databases at once to create the desired final product. The design of the database module can be found above in figure 3.

Like the locations module, the database module connects via a terminal connection and webserver connection with the run.js module or user. The webserver unit and terminal unit of the module will via a program flow connect with the check config component. This component will check the settings as stated in the configuration file. This configuration file will be created by the user or can be generated via another program. In chapter 4 we will describe the precise layout and information which has to be stated in the configuration file.

Once the configuration file is checked and implemented, the check config unit sends signals to the create children unit. This unit creates child processes for the transformation of the to be processed database files, as stated in the configuration file. Every process will be handled separately. By creating child-processes, the system can handle multiple processes at once and will still be able to fetch requests. Therefore, this enhances both the sustainability and efficiency of the system.

Finally, when the child processes are finished and all the data is processed, the log / error check unit fetches all the signals send by its child processes and sends these to the TTY terminal unit and webserver API unit. When any errors have occurred, the error-codes will be added by the log / error unit.

Finally, all the data is gathered and information (signals and log) is send back to the user or run.js. And via the web connection to the other connected modules or API connection.

3.2.3.1 Database helper module

The design of the database helper module can be found in figure 3.

The database helper module will handle the individual processes within the file. It will check for duplicates and/or errors and save the made alterations in a datafile. The process will start with a test of the connection, to make sure the processes can be started. First via a direct insert method the process records are created and the data is checked for duplicates and errors before being inserted into the datafile. It will also be possible to create a connection with an external database to directly handle the process from an external database. The types of database management systems the system can connect with are described in section 2.5. Finally, the altered and checked data is sent back to the parent process to be merged and checked for errors one final time.

3.2.4 Webserver module and web client

The webserver module can also be seen in figure 1. Although clearly visible in the layout of the system, the webserver module is not considered one of the main modules of the system. The sole purpose of the webserver module is to host the webserver and web client. From there the web-client can connect with the locations and database modules.

The web client is a react app connecting to the API's of the location and database modules. By implementing a web interface, it will be easier for users to use and interact with the system.

3.2.5 Support modules

As can be seen in figure 1 a unit called Support Modules is connected to the system as a whole via function call connections. These modules are part of the system, but created in separate smaller modules to support the system. Mostly these modules consist of functions which have multiple applications throughout the system. Therefore, separating them from the major modules and making them operate individually, is much more efficient and increases the modularity and sustainability of the system. All the support modules and their functions will be discussed more in depth in chapter 4.

4 Implementation

In the following chapter we will discuss the implementation of the design as mentioned in Chapter 3. We will discuss per part of the design, the implementations made to make the system as a whole and its parts individually work. Furthermore, we will also discuss a couple of user related aspects of the design, such as the different types of commands, the reported logs, warnings and errors and the initialization and working of the configuration file. All this information is needed to properly work with and understand the system.

4.1 Terminal interface

The terminal interface will be the communication channel between the user and the system. In the following section we will discuss the types of commands, arguments and logs a user can use and receive to communicate with the system. A detailed description on how to set-up the use of the application and how to use the application can be found in appendix A and B or in the files configuration and set up guides.

4.1.1 Commands

Commands being used within the application.

exit	Closes a process.
mute [1, 2, ..., n]	Makes it possible to mute the logs of helper processes.
process filepath	Lets a file being processed by the system, the user needs to define the filepath to the file that needs to be processed.
[verbose 0 1 2 3 4]	Change the type of log a user wants to receive

When using run.js to run the whole system it is necessary to add modules to commands:

run|database|locations|webserver [command]

Commands used when starting the application

node run|database|locations|webserver Start the application

Arguments:

-[verbose 0 1 2 3 4]	Choose log type.
-[noColor]	Turn log colors off.
-[noFormat]	Turn log format off.
-[customcfg filepath]	Turn on own configuration settings.

4.1.2 Log

We differentiate between 5 log types: (used with the command verbose [])

- 0 : ERR** Error messages.
- 1 : WARN** Warning messages.
- 2 : INFO** Information messages.
- 3 : DEBUG** Debug messages.
- 4 : DATADMP** Data print messages.

When defining the type of log using the verbose command, all message types with lower number will also be shown in the outputted log.

4.2 Command and Data flows

When using the system, commands and data flows throughout the system. To give a better understanding as how these program flows operate within the application, we will give an overview of the general command flow, the API command flow and the dataflows of the two main modules. This will provide a clear overview of how the commands and data are being handled by the system.

4.2.1 General command flow

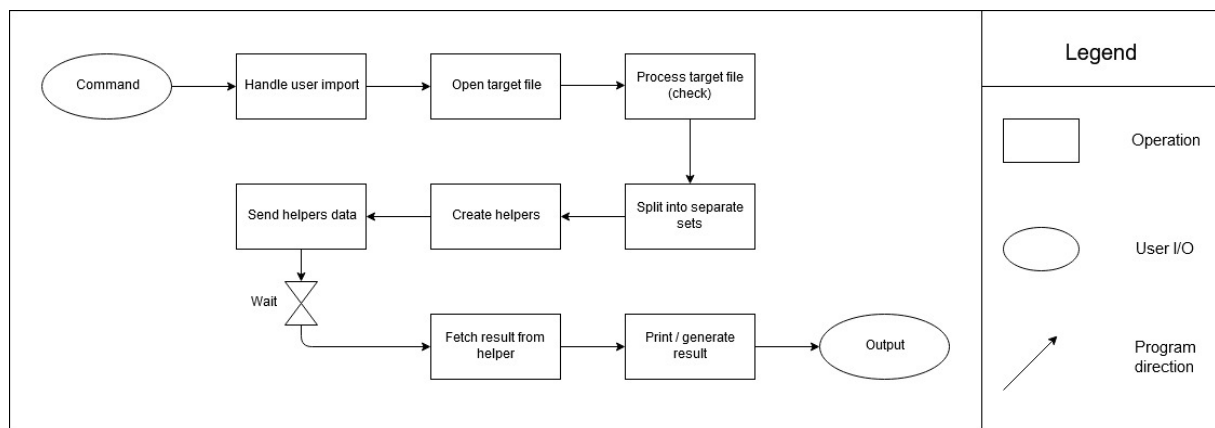


Figure 4 - General command flow overview

In Figure 4 above is the general command flow given. Every command will be handled by the system in the following way, regardless of the functionality of the system used by the user. Using this standard flow creates a clear process and makes it easy to find errors and tackle the right processes.

4.2.2 API command flow

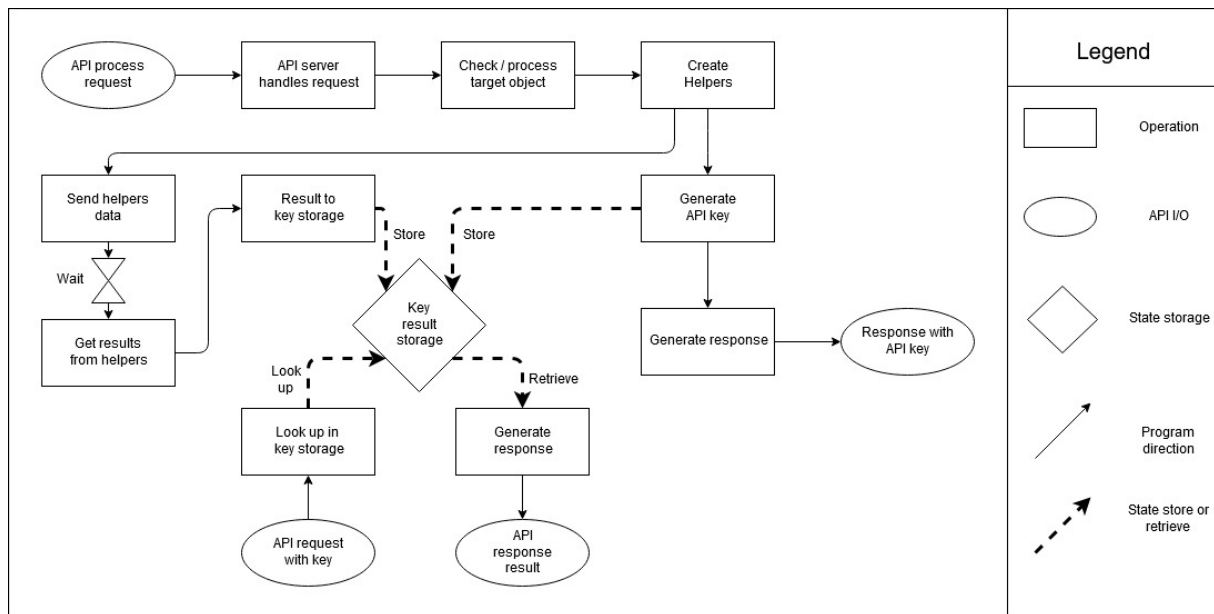


Figure 5 - API command flow overview

In figure 5 the overview of the API requests command flow is given. It is possible to perform operations with the system via API process request. As can be seen in figure 5, when performing an API process request, while creating the helpers, an API key is generated. This key is then stored in the key result storage, where eventually the results of the operations will also be stored to be retrieved by the user. The user can view the progress and result of the performed operation using the given API response key as generated when creating the request. As can be seen in figure 5, when using the API key to request results, a look up of the key storage is performed. When the results are finalized by the underlying processes, these will be given as a response. If the results are not yet stored in the key result storage, a wait message will be given as a response. The user will be able to reperform this until the results will be given.

4.2.3 Database data flow

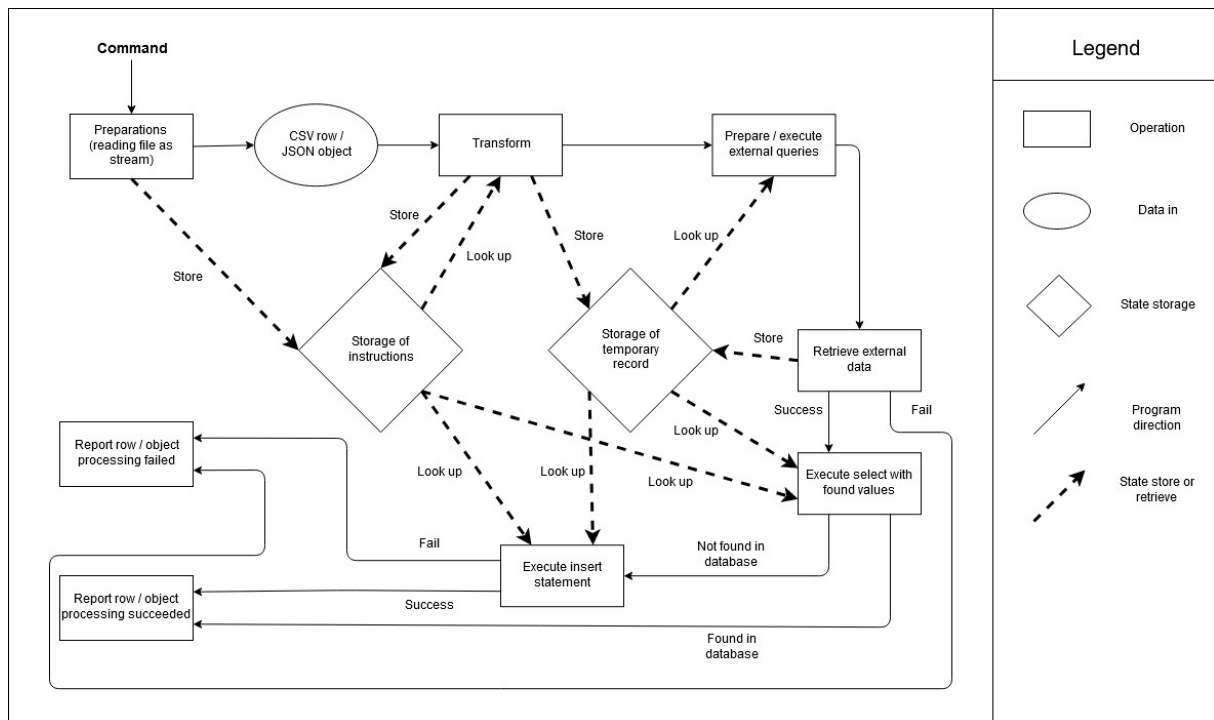


Figure 6 - Database module data flow overview

In Figure 6 above, the data flow within the database and database helper modules is stated. As can be seen the modules make use of two types of storages for records. The first one stores the type of instruction given in the configuration files; the second storage is of a temporary record which stores the modified data along the process. If the process fails to retrieve the external data from the database it will give out a message that the row or object processing has failed. If the external data could be retrieved, the select statement with the proper instructions and saved record is being executed. When found in database, the process is finished and a success message will be given. In the case that the values could not be found in the connected database an insert statement with the given instructions and temporary data will be performed. In case this process succeeds or fails a corresponding message is given.

4.2.4 Locations data flow

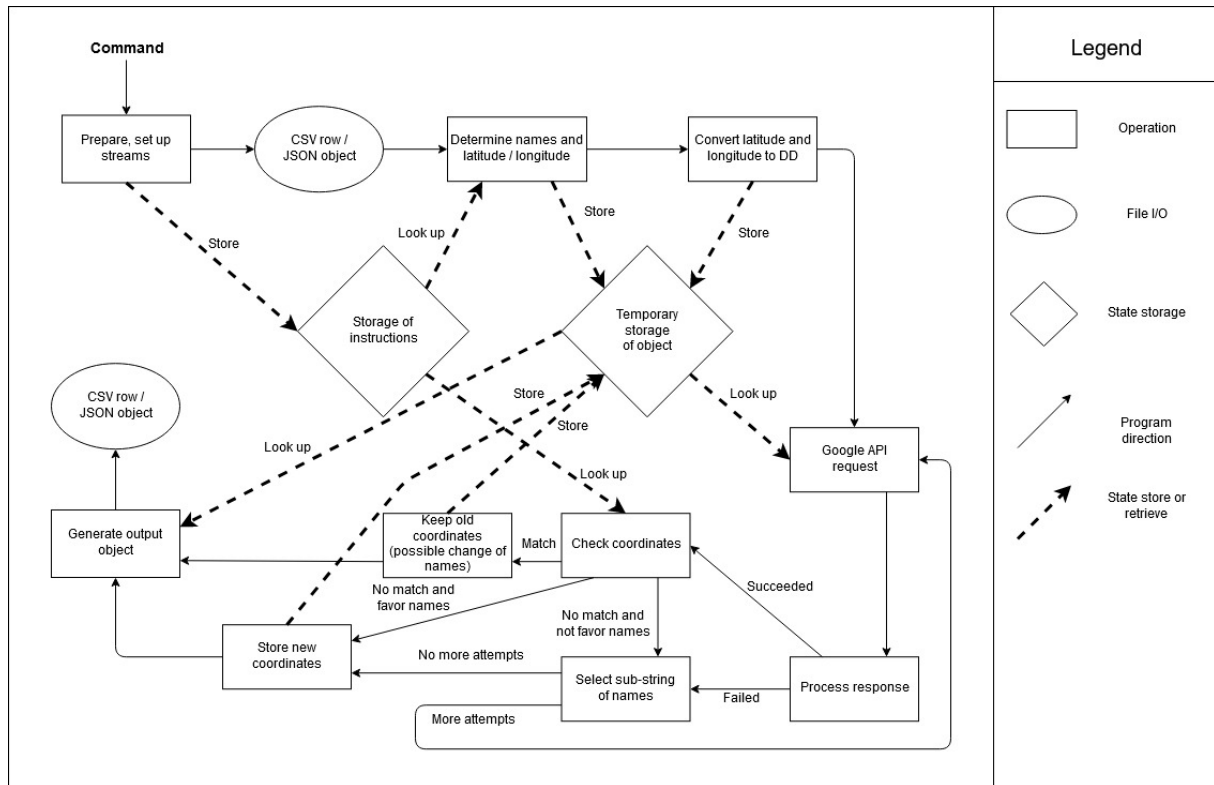


Figure 7 - Locations module data flow overview

The dataflow of the locations and location helper modules is given in Figure 7. As can be seen, this process also uses two types of temporary storages. The first one stores the set of instructions as stated in the configuration file. The second storage consists of the temporary storage of the object to be added to the output file. The temporary storage of the object constantly alters during the data flow process when information is added or altered.

The bottom part of figure 7 describes the process after a google API request has been made. A couple options arise, if the process response failed. Either another attempt is made with a less detailed sub-string of the given name or no more attempts are made and the output is stored as is. If the process response succeeds, the coordinates are checked. In case of a match the old coordinates are stored and an object is generated.

If the process response succeeds but does not match with the provided coordinates 2 possible options are provided, depending on the settings (FavorNames) as provided by the user in the configuration file.

If FavorNames is set to false, a smaller subset of the provided location information is taken and used for a new Google API request. This will be repeated until a match is found or it is no longer possible to take smaller subsets. The last used location information subset will be stored as the used name.

When FavorNames is set to true, the location information set in binding. Thus, in case of a mismatch the new coordinates are saved together with the old location information and put into the output-file.

4.3 User Instructions

In this subsection we will further elaborate on the use of the previously outlined modules and working of the system as a whole. We will make use of examples to provide an overview of the working of the application from a user perspective. The section will be divided into two parts, one to describe the process to check and improve geographic coordinates and another for the importing information into existing databases.

4.3.1 Locations module

A couple of files need to be provided by the user in order to be able to process the datasets. These files are:

- The input datasets in csv or json format, containing at least one field denoting the area information.
- A configuration file containing the settings to process the input datasets. (The description of this file can be found in Appendix B. The process as to how to write such a file can be found in Appendix C).
- (In case of an API request, one needs to define a JavaScript file containing a script for an API request. The API endpoints and process are described in both Appendix A and C.

A detailed example of how to perform the whole process can be found in Appendix C.

4.3.2 Database module

To be able to perform operations on datasets and insert these into the coupled database systems a couple of files need and connections need to be made:

- An input file containing the information which needs to be inputted in the connected database system.
- A configuration file containing the settings to process the input dataset and insert it into the connected database system. (A description of this file can be found in Appendix B. An example of how to write this file can be found in Appendix C.)
- A connected database system.
- (In case of an API request, one needs to define a JavaScript file containing a script for an API request. The API endpoints and process are described in both Appendix A and C.

A detailed example of how to perform the whole process can be found in Appendix C.

4.4 Main modules

In this subsection we will discuss the main modules, which control and operate the system and its core functionalities. For every module we will first discuss its core operation, followed by its underlying functions, the internal and external connections and support modules it uses (a combination of library support modules as well as self-designed). In the database and locations module this is followed by an explanation of the user configuration files.

4.4.1 Run module

4.4.1.1 *Operation*

The Run module is the heart of the system. This module controls the other modules and makes sure they keep running. Furthermore, it handles the information and commands channels between the modules and the terminal interface.

4.4.1.2 *Functions*

typeStrToNr transforms log type to number.

processChildMessage processes the messages send by child processes.

checkOpenClients returns if there are any clients open to be used by the system.

crashHandle returns error messages and handles the crash handling of the other modules. This can be set up to let modules crash alone or let the whole system shut down together.

database create client process for the database module and hooks their events

webserver creates the client process for the webserver module and hooks their events

locations create the client process for the locations module and hooks their events

closeAllClients is a function to close all the client processes.

shutdownTimer checks every increment of time for a possibility to shut down.

parseUserInput is a function to parse user input and translate to usable signals for the system.

4.4.1.3 *Connections*

The connections within the module are process connections. When communicating with other modules and the user, the run module handles command channels, log channels to and from the user and the other modules and the creation of child processes.

4.4.1.4 *Support modules*

The run module makes use of the **args**, **interfaces** and **child processes** support module.

4.4.2 Database module

4.4.2.1 *Operation*

Setting up the process for the database transformations by controlling the user configuration file, testing of the connection and creating and handling of child processes.

4.4.2.2 Functions

parseMessage parses the messages from the parent process (run module). Only in case if run as client process.

parseUserInput parses the messages given by the user, in case of both stand-alone and when using the run module.

afterInit handles the actions after the initialization of the process configuration, and handles errors.

testConnection tests the connection with the external database.

processConfFile processes the user configuration file.

handleAPIInstruction handling of the API requests made via the API support module.

4.4.2.3 Connections

Internal connections within the module are process flows and log. External connections are create-child-process connections both incoming and outgoing. http connections, command and log channels.

4.4.3 Support module

The support modules used by the database module are: **querystring**, **args**, **API**, **config**, **exit**, **helper**, **interfaces** and **typehandler**.

4.4.3.1 User configuration

The database configuration file is a file made by the user to streamline the database transformation process. The readme file and configuration syntax guide can be found in respectively Appendix A and B. The file contains a set of datasets, defined by the square brackets []. Each set (defined by curly brackets { }) contains a database item and a data set.

The database item contains information regarding the database settings:

- “host”: the server address with which the system needs to connect. (text)
- “port”: the port the system needs to connect with. (num)
- “user”: user login information (text)
- “password”: password login information (text)
- “database”: database type (text)
- “ssl”: this is another item containing “ca” where the ssl certification file needs to be stated. (text)

The data item contains sets with the following information:

- “table”: the tablename of the table in which the information needs to be stored.
- “id”: the tableID of the table in which the information needs to be stored
- “method”: is an item containing the Booleans:
 - “insert”:
 - “update”:
 - “check_first”:
- “source”: is an item containing information regarding the sourcefile
 - “file”: a path to the file which contains the to be manipulated data
 - “type”:
 - “delimiter”: what space delimiter is used in the sourcefile

- The last item in data the “mapping” item which again contains a set with a (number of) mapping item(s), which contains:
 - “source”: the name of the sourcefile

4.4.4 Database helper module

4.4.4.1 *Operation*

Links input to the transform support module and creates SQL code to be inputted in the external database. The module does this by using a subset of the user configuration file as its settings.

4.4.4.2 *Functions*

parseMessage handles messages received parent process.

stringDBTypeToNumType transforms stringtype into a numbertype, for internal use.

chooseNullStmt chooses the right statement from a predetermined list of statements according to a possible occurrence of NULL in the values.

giveConnectionObject generates an object for the creation of a connection to an external database.

checkAllFinished checks if all records are processed.

createConnections creates database connections.

prepareConnections prepares database connections.

direct_insert_flow general program flow for direct insert method of database insertion.

executeExtStmts execute select statements on external databases, for getting external values.

executeSelStmt execute select statements to input database.

executeInsStmt execute insert statement to input database.

checkDone checks if all records are processed.

printResults creates log messages with the results.

checkTargets checks if the configuration file is valid.

4.4.4.3 *Connections*

Internal connections are a log stream and process flow. External connections consist of child process connection, log stream, data stream and database connection.

4.4.4.4 *Support modules*

The support modules used by the database helper module are: **mySQL2**, **monetDB**, **MariaDB**, **csv_parse_stream**, **filesystem**, **args**, **databasesqlgen**, **exit**, **interfaces**, **transform** and **typehandler**.

4.4.5 Location module

4.4.5.1 Operation

Interprets the user configuration file and splits up the commands accordingly for the to be made helper modules.

4.4.5.2 Functions

processConfFile processes the user configuration file.

processDataset processes the dataset part of the configuration file and creates helper process accordingly.

ProcessMessage processes the received messages.

parseUserInput parses user input given and executes accordingly.

handleAPIInstruction callback function for process specific API handling.

configCallback in case of error with configuration file, report will be given.

4.4.5.3 Connections

Internal connections within the module are process flows, signals and log. External connections are create-child-process connections both incoming and outgoing. http connections, command and log channels.

4.4.5.4 Support modules

The support modules used by the location module are: **filesystem**, **API**, **args**, **config**, **exit**, **helper**, **interfaces** and **typehandler**.

4.4.5.5 User configuration

The location user configuration file is a file made by the user to streamline the data processing. The Readme file Configuration Syntax Guide can be found in respectively Appendix A and B. It contains a datasets item which is initiated using square brackets []. Within the datasets item, multiple datasets can be defined by the user. Each dataset be defined by braces { }. Within a dataset a number of settings need to be defined:

- “source”: the path to the file containing the data that needs to be checked or transformed. (text)
- “target”: the path to the file in which the altered data will be stored. (text)
- “lat” and “lng” or “latlng”: needs to contain the column name(s) in which the latitude and longitude are stated. (either the “lat” and “lng” column names need to be given, or when given in one field the “latlng” column name need to be given.)
- “places”: contains another set with all the column name(s) of the columns containing information regarding the location (e.g. Country, City, Area)
- “margin”: contains the margin of error, the algorithm will take into account when checking the data.

A user can define the below mentioned Booleans, when left out the default setting of all will be false.

- “includeOld”:
- “includePlaceID”:
- “includeCorrected”:

4.4.6 Location helper module

4.4.6.1 Operation

Parsing of the input file and correcting and adding of GPS location data to the output file.

4.4.6.2 Functions

process_config processes the part of the configuration file needed for the specific child process.

create_csv_header_obj is a constructor for default header object, used by CSV input and output files.

get_csv_headers_from_file import csv header information from external import file.

get_json_headers_from_file import json header information from external import file.

finish is the callback function for the general process execution of the child process and stopping the child process.

run initiates the child process and makes sure the pipeline and actual location checks are executed.

4.4.6.3 Connections

The internal connections of the module consist of process flow and data streams. External connections consist of create-child-process connection, log stream, http connection with the google API and data streams.

4.4.6.4 Support modules

The support modules used by the location helper module are: **csv_parse**, **csv_stringify**, **args**, **exit**, **interfaces**, **typehandler**, **coordinates**, **JSONstream** and **JSONstringify**.

4.5 Support modules

In the following subsection we will discuss the support modules present within the system. These functions can be referenced and used by the main modules and their helpers. For all the support modules, we will first discuss the goal of the module and then elaborate on the subprocesses it consists of.

4.5.1 API Module

Hosting of a webAPI and making it possible for modules to interact with each other and external applications.

4.5.1.1 Subprocesses

addInterfaces connects interfaces module to the API module.

SetSSL sets the SSL key and certificate and restarts the API server if active.

SetNoSSL force disables SSL, can be used both as not setting a SSL in the first place as well as disabling SSL per user request and restarts the API server if active.

setPort will parse the given port and saves this and restarts the API server if active.

getPort returns the port, the API server will run at.

checkPort will check if a certain port is free to use.

startServer will start the API server.

setAddress will set the address at which the API server runs.

getAddress will return the address the server is running at as [protocol://host-address:port].

closeServer closes the server.

restartServer restarts the server.

4.5.2 Args Module

Interpreting and processing of the arguments of the system when detected and also generate a list of argument for the helpers.

4.5.2.1 Subprocesses

setExpected defining the actions for certain arguments. The parameter contains an object which consists of name-function pairs, where the name describes the argument and the function is the action to be executed when the argument is encountered.

setHelperExpected defining the actions for certain arguments within helper functions. The parameter contains an object with name-value pairs, where the name is the argument and the value the number of reserved value spaces behind the initial argument.

getAllArgs returns a list of all the arguments.

getHelperArgs returns a list of all the arguments filtered for a helper process.

addHelperArgs insert a certain argument to the helper argument list.

clearHelperArgs deletes all helper arguments from the list.

removeStripes removes the addon stripes for arguments. To be used internally only.

checkArgs controls the arguments of the process, and checks if present in the expected arguments list and executes the function associated with the argument.

4.5.3 Config Module

Interpreting the process configuration files and adjusting and checking the settings accordingly.

4.5.3.1 Subprocesses

setSpecialProps sets the module specific properties to be expected in the config file.

addInterfaces connects interfaces module to the config module.

parseProperty parses the properties as stated in the user configuration file to be used by the program.

openFile opens a process configuration file with a certain filename.

4.5.4 Databasesqlgen

Generating of SQL code from given properties.

4.5.4.1 Subprocesses

typify changes the datatype of a variable to the right datatype.

query_select_simple creates a simple select query using string table (table to access in the database), array items (the items in the table), nulls (a bit array encoded as a number), parameters (the where clause of the SQL functions).

query_select_simple_values creates a simple select query using string table (table to access in the database), array items (the items in the table), array values (values of the items) and optionally array types (types of the values).

query_select_all_simple creates a simple select all query using string table (table to access in the database), parameters (the where clause of the SQL functions) and nulls (a bit array encoded as a number).

query_select_all_simple_values creates a simple select all query using string table (table to access in the database), parameters (the where clause of the SQL functions), array values (values of the items) and optionally array types (types of the values).

query_insert_simple creates a simple insert query using string table (table to access in the database) and array items (the items in the table).

query_insert_simple_values creates a simple insert query using string table (table to access in the database), array items (the items in the table), array values (values of the items) and optionally array types (types of the values).

query_update_simple which contains an update query where values will be escaped by the database handler. The query uses string table (table to access in the database), array items (the items in the table), where (the where clause of the SQL functions), nulls (a bit array encoded as a number), relations (in what way the values compare to each other)

query_update_simple_values which contains an update query where values will be escaped by this module. The query uses string table (table to access in the database), array items (the items in the table), array values (values of the items), parameters (the items of the where clause of the SQL functions), parameter values (the values of the where clause of the SQL functions), relations (in what way the values compare to each other).

4.5.5 Exit Module

Shutting down processes and helper processes.

4.5.5.1 Subprocesses

addInterface connects interfaces module to the exit module.

addHelperModule connects the helper module to the exit module.

exitProcess is a function to shut down processes, using an exit code and timer (to create a delay for output to flush).

exitProcess_unbinded is the same as exitProcess while not relying on its own properties for execution. Used for standalone processing.

4.5.6 Helper

Creation of child processes, and handling and sending messages to child processes.

4.5.6.1 Subprocesses

muteHelper mutes a helper with a certain ID.

setHelperFile sets the file to be executed as helper process.

checkCallbacks checks if certain callbacks are set

addInterface connects interfaces module to the helper module.

createHelper creates a helper process.

haltHelpers gives all helper processes a signal to stop.

killHelper forces a certain helper process to stop.

getActiveHelpers returns a list of the active helper processes.

4.5.7 Interfaces Module

Handling the generation and sending of messages. Making them the right format, message, color etc.

4.5.7.1 Subprocesses

setMaster saves if the process is activated by the user or is initiated by another process.

disableFormat turns of the formatting of the log stream.

setDisableColor turns of the color coding of the log stream.

setProgramColor sets a certain color in the log stream.

setProgramTag sets the name of the used module in the log stream.

log_dd/d/i/w/e couples the log message and message type and logs a message.

requestProperty sends request to the parent process for a certain property of a certain module.

4.5.8 Transform Module

Handles the transformation of database file input.

4.5.8.1 Subprocesses

connectInterfaces connects the interfaces to the transform module.

transformSource transforms a data input into another data type according to the given array.

4.5.9 Typehandler

Determining, controlling and processing of the type of variables.

4.5.9.1 Subprocesses

check_defined checks if the object, array, string, number or Boolean is defined.

now returns current date and time.

time returns certain time.

date returns certain date.

datetime returns certain date and time.

integer returns integer.

hex returns a hexadecimal.

float returns a decimal number.

string turns the given input into a string.

bool transforms input into a Boolean type.

check_type checks the data type of a certain input.

create_copy_of_object creates a literal copy of the object, since objects are passed by reference.

set_default_reference_object sets object to which another object can be referenced to.

check_similar_object compares objects by properties, to see if they are the same data type. If no reference defined, it uses the default_reference_object.

check_equal_object check if objects are of similar type and then if they are equal in value. If no reference defined, it uses the default_reference_object.

fill_object_gaps fills gaps in the object by filling it using the reference, note, this does not trim or expand the object. If no reference defined, it uses the default_reference_object.

expand_object_to will add missing properties and set unset properties. If no reference defined, it uses the default_reference_object.

trim_object_to will delete properties from object if they are not found in the reference. If no reference defined, it uses the default_reference_object.

sanitize_html removes HTML tags from the inputted string.

4.5.10 Coordinates Module

Reading and transforming of GPS coordinates data.

4.5.10.1 Subprocesses

getCoordFromString returns coordinates from an input string.

convertDMStoDD when DMS type of coordinate notation are found, they are then converted to DD type of coordinate notation, for separated latitude and longitude type of notation.

convertsingleDMStoDD when DMS type of notation is delivered in a single string, this function splits the input data into two separate strings. And converts to DD type of coordinate notation.

4.5.11 JSONstream Module

This module converts stringified JSON to a stream of JavaScript objects. Only supports a simple JSON array.

4.5.11.1 Subprocesses

createJSONstream creates a JSON stream object.

transformFunc executes the transformation of stringified JSON to JavaScript objects.

4.5.12 JSONstringify Module

Convert a string of JavaScript objects to a stringified JSON stream.

4.5.12.1 Subprocesses

createStringifyStream creates a stringified JSON stream object.

objectToStr converts a JavaScript object to a stringified JSON stream.

transFunc parent function of the objectToStr function. And prints start of array if needed.

flushFunc prints end of array.

5 Experiments

In the Experiments section we will discuss the tests we have performed to further substantiate the design choices and implementations as stated in the previous two sections. First, we will deliberate on the support modules, for which we have created validation tests to check the working of each of these modules. Following, we have created efficiency tests for both the locations and database modules. These efficiency tests will test the speed and accuracy of the application for different sample sizes and percentages of correct data.

5.1 Support modules validation tests

5.1.1 Interface validation test

The interfaces module makes sure all the messages to the terminal are communicated to the user in a clear manner. The validation test performed for the interfaces module, tests if with the proper settings the right information is being shown.

The performed test can be found in Appendix C.1, the results are shown below:

```

1) Show all
2) Show ERR message:
3) [2019-07-27|16:00,15.675|TestProgram] ERR: This is an error
4) Show WARN message:
5) [2019-07-27|16:00,15.676|TestProgram] WARN: This is a warning
6) Show INFO message
7) [2019-07-27|16:00,15.676|TestProgram] INFO: This is an info message
8) Show DEBUG message
9) [2019-07-27|16:00,15.676|TestProgram] DEBUG: This is a DEBUG message
10) Show data dump
11) [2019-07-27|16:00,15.677|TestProgram] DATADMP: Variable: datadump =
    ["data","data","data"]
12) Below there should not be any DEBUG or data dump:
13) Show ERR message:
14) [2019-07-27|16:00,15.677|TestProgram] ERR: This is an error
15) Show WARN message:
16) [2019-07-27|16:00,15.677|TestProgram] WARN: This is a warning
17) Show INFO message
18) [2019-07-27|16:00,15.677|TestProgram] INFO: This is an info message
19) Show DEBUG message
20) Show data dump
21) Below there should only be an error
22) Show ERR message:
23) [2019-07-27|16:00,15.678|TestProgram] ERR: This is an error
24) Show WARN message:
25) Show INFO message
26) Show DEBUG message
27) Show data dump
28) Show all messages without color.
29) Show ERR message:
30) [2019-07-27|16:00,15.679|TestProgram] ERR: This is an error
31) Show WARN message:
32) [2019-07-27|16:00,15.679|TestProgram] WARN: This is a warning
33) Show INFO message
34) [2019-07-27|16:00,15.679|TestProgram] INFO: This is an info message
35) Show DEBUG message
36) [2019-07-27|16:00,15.679|TestProgram] DEBUG: This is a DEBUG message
37) Show data dump
38) [2019-07-27|16:00,15.679|TestProgram] DATADMP: Variable: datadump =
    ["data","data","data"]

```

As can be seen, line 6 to 16 should show all message types with color coding. Line 17 to 25 should show all message types except for the DEBUG and datadump message, with color coding. Line 26 to

32 should only show a color-coded ERROR message. And line 33 to 43 should show all messages without color coding.

As can be seen in the above shown output, all settings work as expected. Therefore, we can conclude that the interfaces module works as expected.

5.1.2 Typehandler validation test

The typehandler module is used to check for similarities between objects. The module compares two objects and the items of the second object should also be present in the first object. To validate the working of this module a test was created, to see how the module would react to certain inputs. Also, an expected answer would be given, if both outputs would be the same, the module works properly.

The code for the typehandler validation test can be found in Appendix C.2. The test resulted the following output:

```
Expect output true:
true
Expect output false (failed on key count):
false
Expect output false (failed on not having required keys):
false
Expect output true (original may have more properties than reference):
true
Checking copy and equal
Copying {"a":"nee","b":"ja"}
Expect this obj to be the same: {"a":"nee","b":"ja"}
Expect compare to be true
true
```

The first comparison should result in true as the second object contains items of the same type as are present in the first object. The second and third compares should evaluate to false for respectively containing more items than the object being compared to and not having the same properties as object being compared to. The fourth instance compares a bigger object to a smaller one, which evaluates to true because the second object only contains items which are present in the object it compares to.

Also, the functions copy and equal result in similar objects and work properly.

Thus, as can be seen above, the typehandler module works properly and gives the expected results.

5.1.3 JSON stream and stringify validation test

Apart from the CSV node module which is an implemented module of the Node.js system, the application needed a similar module for the use of JSON input files. Therefore, a similar module was created to transform JSON streams to a string of JavaScript objects and the other way around. To validate the working of these modules we created a simple test which would transform a json test file into a stream of JavaScript objects which would then be transformed into a new json file.

The code for the test, as can be found in Appendix C.3, would evaluate into the following results.

testjsonstream.json

```
[
  {
```

```

    "Sample_pk": "SAM000292",
    "Sample Name": "S075",
    "Country": "Argentina",
    "Location": "Arroyo Los Notros",
    "Latitude": -112.6346,
    "Longitude": 0,
    "Amount": 461
  },
  {
    "Sample_pk": "SAM000298",
    "Sample Name": "S081",
    "Country": "Argentina",
    "Location": "Camino a Que?i",
    "Latitude": -111.8066,
    "Longitude": 0,
    "Amount": 516
  }
]

```

json_stream_output.json

```

[
  {
    "Sample_pk": "SAM000292",
    "Sample Name": "S075",
    "Country": "Argentina",
    "Location": "Arroyo Los Notros",
    "Latitude": -112.6346,
    "Longitude": 0,
    "Amount": 461
  },
  {
    "Sample_pk": "SAM000298",
    "Sample Name": "S081",
    "Country": "Argentina",
    "Location": "Camino a Que?i",
    "Latitude": -111.8066,
    "Longitude": 0,
    "Amount": 516
  }
]

```

The expected result would be for both files to be exactly the same. As can be seen above and in the files, both files contain the exact same information. This means that both the json stream and stringify modules work properly.

5.1.4 DMS to DD transformation test

The following validation test was to check if the transformation of DMS to DD type GPS notation within the system performs correctly. We have created a test with 8 different notations, which all have to translate to the same coordinates. These are also the only types of coordinate notations the system will be able to handle.

The performed test can be found in Appendix C.4. The test delivered the following results:

```
coord_1 (52°14'36",5°38'3") evaluates to: 52.24333333333333,5.6341666666666666
coord_2 (-52°14'36"S,5°38'3") evaluates to: 52.24333333333333,5.6341666666666666
coord_3 (52°14'36",-5°38'3"W) evaluates to: 52.24333333333333,5.6341666666666666
coord_4 (-52°14'36"S,-5°38'3"W) evaluates to: 52.24333333333333,5.6341666666666666
coord_5 (52°14'36"N,5°38'3"E) evaluates to: 52.24333333333333,5.6341666666666666
coord_6 (-5°38'3"W,52°14'36"N) evaluates to: 52.24333333333333,5.6341666666666666
coord_7 (5°38'3"E,-52°14'36"S) evaluates to: 52.24333333333333,5.6341666666666666
coord_8 (52.24333333,5.63416667) evaluates to: 52.24333333,5.63416667
```

The first 7 notations are some type of the DMS notation system. The system should be able to check if the E/W or N/S notation comes first and should be able to handle negative values. Also, it should not matter if the latitude or longitude notation is ended with N/E/S/W or not.

As can be seen above, all coordinates evaluate to the same location. Therefore, we can conclude that our DMS to DD transformation module works as expected.

5.1.5 Database SQL generation validation test

For the database module to operate properly, the SQL database operations need to work as expected. A validation test was created to test the working of these operations. The code for this test can be found in Appendix C.5, the results are stated below:

```
Expected: SELECT id FROM testtable WHERE ( item1 = ? AND item2 = ? AND item3 = ? )
SELECT `id` FROM `testtable` WHERE (`item1` = ? AND `item2` = ? AND `item3` = ?)
Expected: SELECT * FROM testtable WHERE ( item1 = ? AND item2 = ? AND item3 = ? )
SELECT * FROM `testtable` WHERE (`item1` = ? AND `item2` = ? AND `item3` = ?)
Expected: INSERT INTO testtable ( item1, item2, item3 ) VALUES ( ?, ?, ? )
INSERT INTO `testtable` (`item1`, `item2`, `item3`) VALUES ( ?, ?, ? )
```

As can be seen above all the expected commands were also plotted when performing the operations. Therefore, the Database SQL generation module works properly.

5.1.6 Transform validation module

Another essential support module for the database module to operate is the transform module. This module handles all the data transformations stated in the configuration files as given by the user. A test file was created to validate the working of this module. The code of this test file can be found in Appendix C.6. The results to these tests are stated below:

```
transform types initial value float (1.2), expect string, number (whole), number
(comma), boo
lean
a1: 1.2 - string
a2: 1 - number
a3: 1.2 - number
a4: true - boolean
transform regexes, input string abababa, regex = a
a5: a
a6: a
a7: null
transform numbers, modulo div and div-float, input is 10, other input 4
```

```
Expect 0 ( 20%4), 5 (20/4) and 0.5 (20/40)
a8: 0
a9: 5
a10: 0.5
transform char and substring tests: string abcd
a11: abcd
a12: cd
transform condition and value test
expect b and c
b
c
```

The tests with results a1 – a4 are type transformations. A value was given and transformed into the respecting value types. As can be seen all transformations were performed successfully. The test results a5 and a6 both result to a, by picking the regex value from the input string. The test result a7 evaluates to null, as c is not part of the input string.

Also, number transformations are possible: modulo, divide and divide-float. Which results are stated at a8 – a10. With a given value of 20 the respecting operations resulted the expected results.

Test results a11 and a12 transformed a string abcd into respectively a character set abcd and a substring starting at the third character.

Lastly the if statement transformation was tested and the test resulted in the expected results.

With all the parts of the transformation module resulting the expected results, we can conclude that this module works properly.

5.2 Location module tests

5.2.1 Speed test

To test the speed of the location module of the system we have created a set of test files to check the speed of the module in different settings. The test sets used for the testing are 3 sets of different sizes (10, 100 and 1000 input objects) with each 3 sets of different percentages of accurate information (0, 50 and 100 % correct). These settings were tested with both the FavorNames setting set to true and false and tested on 3 different systems (Laptop, PC and Server).

We used these settings to test the influence of both size and accuracy of datasets on the duration of the processing time of the module. By testing the outer limits as well as the exact middle of the possibilities we were able to show the full range of the possible duration. Furthermore, we wanted to test the influence of the FavorNames setting on the duration, because of the possible difference in API requests this function can cause.

An example of the test sets of 10 input objects can be found in tables 1 - 3 below, the test sets of size 100 and 1000 were expanded respectively. The inputs with a longitude of 5.171 are situated in the Netherlands, the inputs with a longitude of 8.250 are situated outside of the Netherlands, but within Europe. All tests were performed using the API module of the system.

latitude	longitude	Continent	Country	expected
51.345	8.250	Europe	Netherlands	false
51.346	8.250	Europe	Netherlands	false
51.347	8.250	Europe	Netherlands	false
51.348	8.250	Europe	Netherlands	false
51.349	8.250	Europe	Netherlands	false
51.350	8.250	Europe	Netherlands	false
51.351	8.250	Europe	Netherlands	false
51.352	8.250	Europe	Netherlands	false
51.353	8.250	Europe	Netherlands	false
51.354	8.250	Europe	Netherlands	false

Table 1 - loc_speed_test_10-0

latitude	longitude	Continent	Country	expected
51.345	5.171	Europe	Netherlands	true
51.346	8.250	Europe	Netherlands	false
51.347	5.171	Europe	Netherlands	true
51.348	8.250	Europe	Netherlands	false
51.349	5.171	Europe	Netherlands	true
51.350	8.250	Europe	Netherlands	false
51.351	5.171	Europe	Netherlands	true
51.352	8.250	Europe	Netherlands	false
51.353	5.171	Europe	Netherlands	true
51.354	8.250	Europe	Netherlands	false

Table 2 - loc_speed_test_10-50

latitude	longitude	Continent	Country	expected
51.345	5.171	Europe	Netherlands	true

51.346	5.171	Europe	Netherlands	true
51.347	5.171	Europe	Netherlands	true
51.348	5.171	Europe	Netherlands	true
51.349	5.171	Europe	Netherlands	true
51.350	5.171	Europe	Netherlands	true
51.351	5.171	Europe	Netherlands	true
51.352	5.171	Europe	Netherlands	true
51.353	5.171	Europe	Netherlands	true
51.354	5.171	Europe	Netherlands	true

Table 3 - loc_speed_test_10-100

Each setting was tested for a total of 5 times per system. The results of those tests can be found in Appendix E.2 – E.7 and the average results are shown in Table 4 below.

averages FNAAN		Laptop	PC	Server
sample size	% correct	duration (ms)	duration (ms)	duration (ms)
10	0%	1399	655	392
	50%	1378	642	394
	100%	1401	638	396
100	0%	12797	2984	3342
	50%	12000	2984	3400
	100%	12092	3032	3383
1000	0%	119436	32070	31868
	50%	122104	30327	32229
	100%	125190	31259	32440

averages FNUIT		Laptop	PC	Server
sample size	% correct	duration (ms)	duration (ms)	duration (ms)
10	0%	2547	905	721
	50%	2044	747	562
	100%	1353	603	398
100	0%	23926	5693	6577
	50%	17594	4326	4889
	100%	11711	2878	3307
1000	0%	246117	59023	68298
	50%	184981	45668	50301
	100%	123495	29334	32556

Table 4 - Average results of the location module speed tests (FavorNames True and FavorNames False)

As can be seen in table 4, there are some clear differences of the speed between the different systems and settings. These differences are visualized in the graphs in figures 8 and 9 below, to give a better insight in the comparison of these numbers.

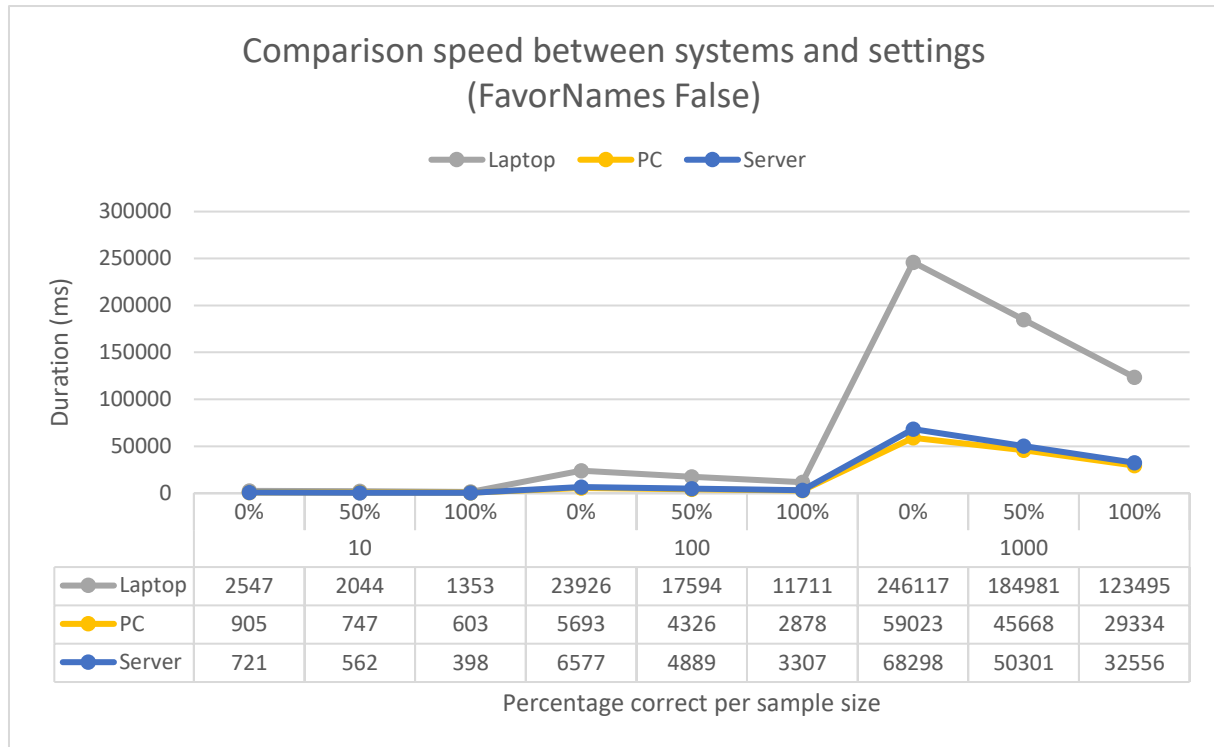


Figure 8 - Comparison speed between systems and settings (FavorNames False)

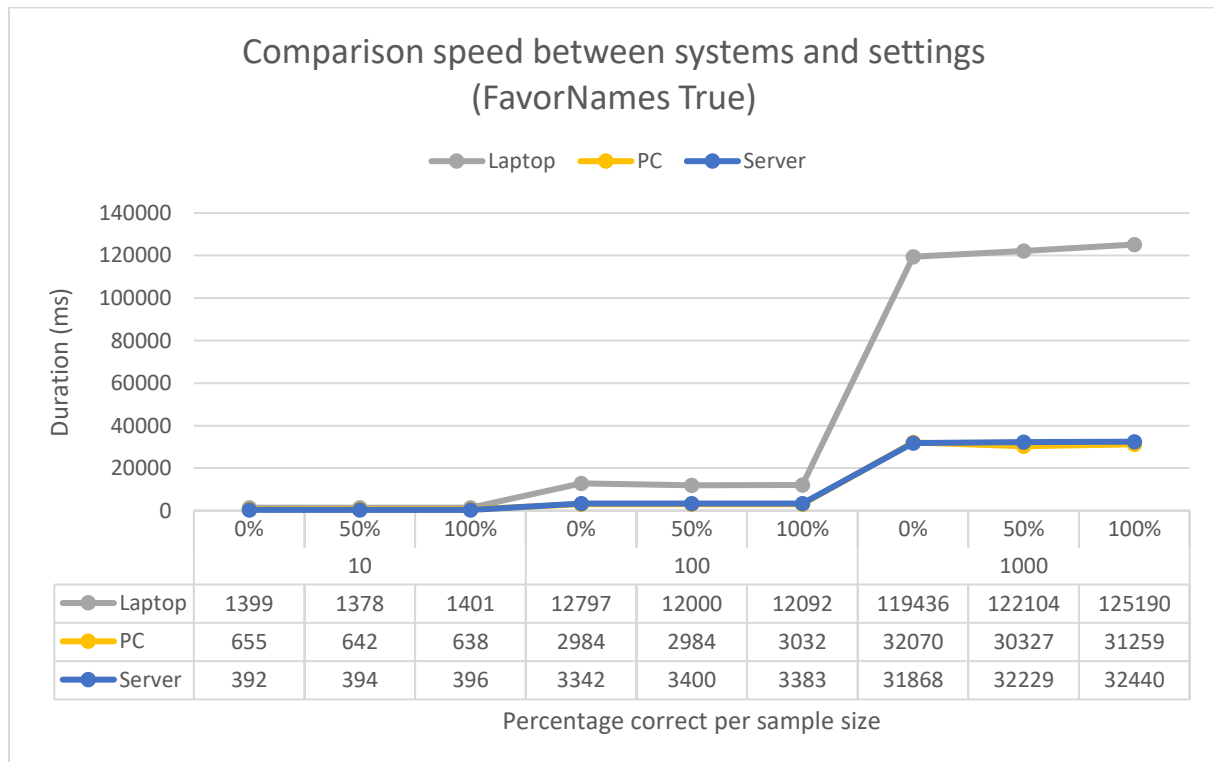


Figure 9 - Comparison speed between systems and settings (FavorNames True)

As can be seen in figures 8 and 9 above there is a clear difference between the performance of the Laptop and the other systems when it comes to the speed of the performed tests. Contributing factors towards this end are the specifications of the systems, which dictates the speed at which the program can perform its functions. These specifications are given in Appendix D. The differences between the speed of the tests performed on the PC and Server are marginal. However, the

performed tests on the server system ran into some complications. Due to the high speed and efficiency of the system, the tests sometimes ran above the limits allowed by the Google Geocoding API. These limits allow for a maximum number of 50 requests per second or 5000 requests per 100 seconds. When performing operations using our system with a large sample size (10000+ data entries), this can result in complications. The system will report this using warning messages, informing the user the query limit of the API has been reached. The user will need to perform the process again, using a smaller dataset to prevent the same error from happening again. Because of the beforementioned problem it is recommended to keep the sample sizes below this threshold or cut inputs in multiple portions.

When looking at differences in specifications between the systems, a clear difference can be seen between the network speed of the laptop test and the other 2 systems. This most likely had the most impact on the clear speed difference when looking at the Laptop performed tests. The reason behind this is the number of google requests that need to be performed, which require sending and receiving data via an internet connection.

Another clear difference between the figures 8 and 9 are the differences within the different sample sizes. When FavorNames is set to true, the speed of the tests is not influenced by the percentage of correct given GPS coordinates. Whereas if FavorNames is set to false, the speed is extremely influenced by the changes in percentage of correct given GPS coordinates. In figures 10 – 12 below the differences between the FavorNames settings are compared for every system.

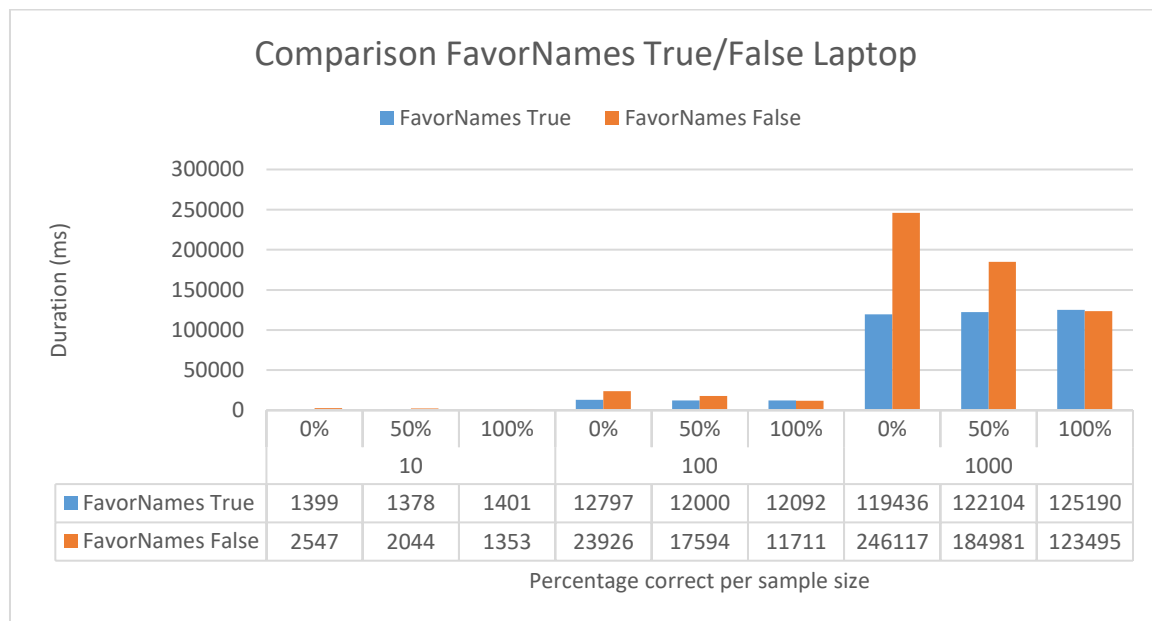


Figure 10 - Comparison FavorNames True/False Laptop

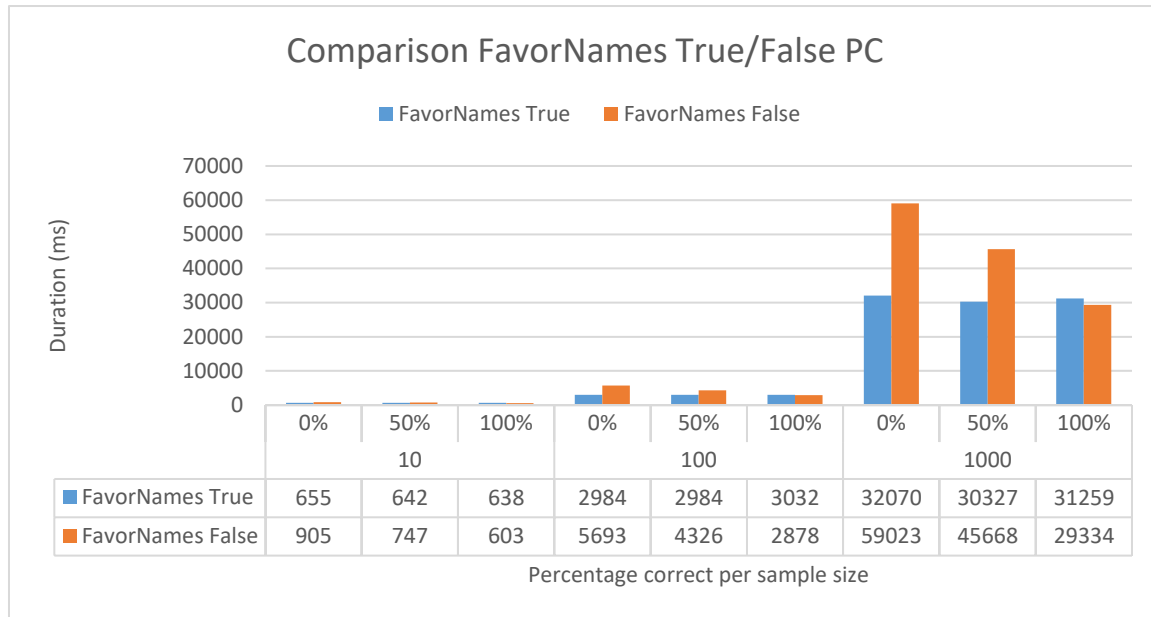


Figure 11 - Comparison FavorNames True/False PC

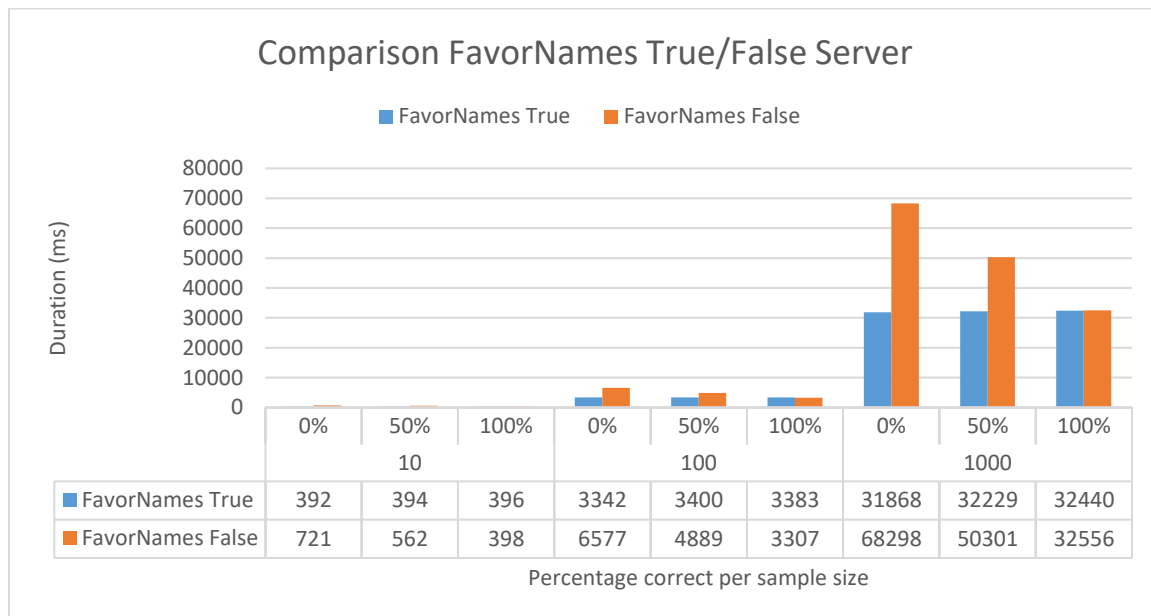


Figure 12 - Comparison FavorNames True/False Server

As can be seen in the figures 10 – 12 above the duration of the operations is approximately constant when the FavorNames is set to true for all given systems. And when FavorNames is set to false the 100% settings are approximately the same as when FavorNames is set to true and respectively the 50% and 0% settings take 1.5 and 2 times as long to finish the test operations. The logical explanation for this can be seen in the raw data in Appendix D. When FavorNames is set to True, the operation performs as many requests to Google as there are data entries in the test file. When FavorNames is set to False, the number of requests depends on the percentage of correct given GPS coordinates and the possibility to zoom out. In our test cases, also the option to zoom out from ‘the Netherlands’ to ‘Europe’ was an option. Thus, meaning that for every false record, the system performed an extra data request to Google. This increased the duration of the system appropriately.

5.2.2 Spelling test

An issue of the system is that the Google Geocoding API preferences American names and cities when given little information or names using a different language. The tests sets given in the tables 5 and 6 below were used to test the complications this could give and if providing more information with the data entries could tackle this problem.

latitude	longitude	country
-24.422	-69.290	Chili
-25.422	-69.290	Chile
37.192	-90.680	USA
36.192	-90.680	US
53.124	5.613	Nederland
52.124	5.613	the Netherlands
41.924	12.492	Rome
41.899	12.497	Roma

Table 5 – Spelling test input (country)

latitude	longitude	continent	country
-24.422	-69.290	South America	Chili
-25.422	-69.290	South America	Chile
-26.422	-69.290	Zuid Amerika	Chili
-23.422	-69.290	Zuid Amerika	Chile
37.192	-90.680	North America	USA
36.192	-90.680	North America	US
35.192	-90.680	Noord Amerika	USA
34.192	-90.680	Noord Amerika	US
53.124	5.613	Europe	Nederland
52.124	5.613	Europe	the Netherlands
53.378	6.606	Europa	Nederland
52.624	5.613	Europa	the Netherlands
41.924	12.492	Europe	Roma
41.899	12.497	Europe	Rome
41.924	12.497	Europa	Roma
41.899	12.492	Europa	Rome

Table 6 – Spelling test input (continent, country)

As can be seen in tables 5 and 6 the combinations of spelling of English and Dutch were used to test if the given GPS coordinates would be transformed by the application or that they would be given as correct (all coordinates are located in the stated areas).

Latitude	Longitude	Name	Old_Latitude	Old_Longitude	Corrected_Lat	Corrected_Lng
37.056	-95.710	Chili	-24.422	-69.290	Yes	Yes
-25.422	-69.290	Chile	-25.422	-69.290	No	No
37.192	-90.680	USA	37.192	-90.680	No	No
36.192	-90.680	US	36.192	-90.680	No	No
39.961	-105.511	Nederland	53.124	5.613	Yes	Yes
52.124	5.613	the Netherlands	52.124	5.613	No	No
41.924	12.492	Rome	41.924	12.492	No	No
36.077	-95.904	Roma	41.899	12.497	Yes	Yes

Table 7 - Results spelling test (country)

Latitude	Longitude	Name	Old_Latitude	Old_Longitude	Corrected_Lat	Corrected_Lng
-24.422	-69.290	South America chili	-24.422	-69.290	No	No
-25.422	-69.290	South America chile	-25.422	-69.290	No	No
-26.422	-69.290	Zuid Amerika chili	-26.422	-69.290	No	No
-23.422	-69.290	Zuid Amerika chile	-23.422	-69.290	No	No
37.192	-90.680	North America USA	37.192	-90.680	No	No
36.192	-90.680	North America US	36.192	-90.680	No	No
35.192	-90.680	Noord Amerika USA	35.192	-90.680	No	No
34.192	-90.680	Noord Amerika US	34.192	-90.680	No	No
53.124	5.613	Europe Nederland	53.124	5.613	No	No
52.124	5.613	Europe the Netherlands	52.124	5.613	No	No
53.378	6.606	Europa Nederland	53.378	6.606	No	No
52.624	5.613	Europa the Netherlands	52.624	5.613	No	No
41.924	12.453	Europe Roma	41.924	12.453	No	No
41.890	12.497	Europe Rome	41.890	12.497	No	No
41.924	12.497	Europa Roma	41.924	12.497	No	No
41.899	12.492	Europa Rome	41.890	12.453	No	No

Table 8 - Results spelling test (continent, country)

As can be seen in the Tables 5 – 8 above, when only using the country (and city) names, the system and Google Geocoding API wrongly altered coordinates to places in the United States. However, when adding a layer of continents to the input information, none of the correct coordinates were wrongly altered to places in the United States. Therefore, we can conclude that the system can wrongly detect places and alter coordinates mistakenly. However, when expanding the given input information, the user can drastically decrease the chance of mistakes by the system. Another solution is discussed in the discussion and future implementations section.

5.3 Database module tests

5.3.1 ID retrieval and uploading test

For the Database module to work properly the ID retrieval and uploading components of the module need to work properly. The following validation test, checks if both components of the database module work properly. The test splits an incoming database into two separate tables, coupled by a similar ID.

name	email
Henk de Vries	woeste_pvver@knobhe.ad
Jan Klaassen	janklaassen@ohyeah.nl

Table 9 - Input table

+ Opties







			id	name
<input type="checkbox"/>	 Wijzigen	 Kopiëren	 Verwijderen	1 Henk de Vries
<input type="checkbox"/>	 Wijzigen	 Kopiëren	 Verwijderen	2 Jan Klaassen

Figure 13 - Database table 1 (name)

+ Opties

			id	email	name_id
<input type="checkbox"/>	 Wijzigen	 Kopiëren	1	woeste_pwer@knobhe.ad	1
<input type="checkbox"/>	 Wijzigen	 Kopiëren	2	janklaassen@ohyeah.nl	2

Figure 14 - Database table 2 (email)

The input coupled to our test file can be seen in table 9, it contains two data entries both containing a name and email address. The output of the test file can be seen in figure 13 and 14. As can be seen, the first table only contains the names and coupled id's as given by the database. The second table contains email addresses, id's and name_id's which are the coupled id's of the first table. This shows that the coupling of id's and uploading of the data is performed successfully.

5.3.2 Speed test

To test the speed of the database module we created multiple test sets. The different test sets consisted of a variation of test sets with 3 different sample sizes and for each of the sample sizes 3 different settings were tested. The sample sizes used in the testing process were CSV files containing 1000, 10000 and 100000 data entries, consisting of an id, name, email and date. Three different setups per sample size were created. One where the entire datafile consisted of different entries, one consisting of 50% duplicate entries and one file containing only duplicate entries (100%).

We used this setup to create an image of the influence of duplicate entries on the processing speed of the module. By using both the outer limits as well as the median, we wanted to show the duration range at which the module operates.

Examples of the different settings are given below in tables 10 – 12, for a total of 10 entries, to paint an image of the testing input files.

id	name	email	date
0	Eulah Walter	Candelario@wade.name	13-5-1991
1	Erik Nienow	Torrance@shaniya.name	18-11-2006
2	Emory Anderson	Arianna@kaitlyn.co.uk	20-7-2009
3	Ms. Katelyn Dickens	Madaline.Sauer@therese.net	31-8-2003
4	Johnson Dare	Darrick_Walter@elena.tv	30-8-2013
5	Mrs. Reggie Mayert	Destinee_Walsh@dane.org	22-10-2012
6	Seamus Mann DVM	Gerry.OHara@ashleigh.me	27-1-1994
7	Dixie Morar	Ruben.Stroman@ezekiel.net	21-7-2015
8	Rosalyn Wolf	Gino@yazmin.me	14-1-2018
9	Dallas Pagac V	Shaina@edwardo.name	1-4-2000
10	Kayli Wyman	Shanna_Graham@alexander.io	6-3-1992

Table 10 - test input for 0% duplicates example

id	name	email	date
0	Eulah Walter	Candelario@wade.name	13-5-1991
1	Erik Nienow	Torrance@shaniya.name	18-11-2006
2	Emory Anderson	Arianna@kaitlyn.co.uk	20-7-2009
3	Ms. Katelyn Dickens	Madaline.Sauer@therese.net	31-8-2003
4	Johnson Dare	Darrick_Walter@elena.tv	30-8-2013
5	Mrs. Reggie Mayert	Destinee_Walsh@dane.org	22-10-2012
1	Erik Nienow	Torrance@shaniya.name	18-11-2006
2	Emory Anderson	Arianna@kaitlyn.co.uk	20-7-2009
3	Ms. Katelyn Dickens	Madaline.Sauer@therese.net	31-8-2003
4	Johnson Dare	Darrick_Walter@elena.tv	30-8-2013
5	Mrs. Reggie Mayert	Destinee_Walsh@dane.org	22-10-2012

Table 11 - test input for 50% duplicates example

id	name	email	date
0	Eulah Walter	Candelario@wade.name	13-5-1991
1	Eulah Walter	Candelario@wade.name	13-5-1991
2	Eulah Walter	Candelario@wade.name	13-5-1991
3	Eulah Walter	Candelario@wade.name	13-5-1991
4	Eulah Walter	Candelario@wade.name	13-5-1991
5	Eulah Walter	Candelario@wade.name	13-5-1991
6	Eulah Walter	Candelario@wade.name	13-5-1991
7	Eulah Walter	Candelario@wade.name	13-5-1991
8	Eulah Walter	Candelario@wade.name	13-5-1991
9	Eulah Walter	Candelario@wade.name	13-5-1991
10	Eulah Walter	Candelario@wade.name	13-5-1991

Table 12 - test input for 100% duplicates example

Files similar to the above pictured situations were created for the respective sizes and settings. These test files were tested using a Laptop, PC and Server making use of the API module of the application. The specifications of the different systems can be found in Appendix F.1. The raw output data of the

tests can also be found in Appendix F.2 – F.4. As can be seen, every setting was tested for a total of 6 times per setting. The averages of all these tests can be found in table 13 below.

averages		Laptop	PC	Server
sample size	doubles %	duration (ms)	duration (ms)	duration (ms)
1000	0%	25219	20287	12340
	50%	15217	13128	12318
	100%	14500	9117	1327
10000	0%	236556	183769	111764
	50%	172022	103409	109734
	100%	136475	74619	4157
100000	0%	3688190	2977667	1134281
	50%	3154705	2435349	1087727
	100%	1364544	736447	16644

Table 13 - Average results database module speed test

The above results are visualized in the graph represented in figure 14.

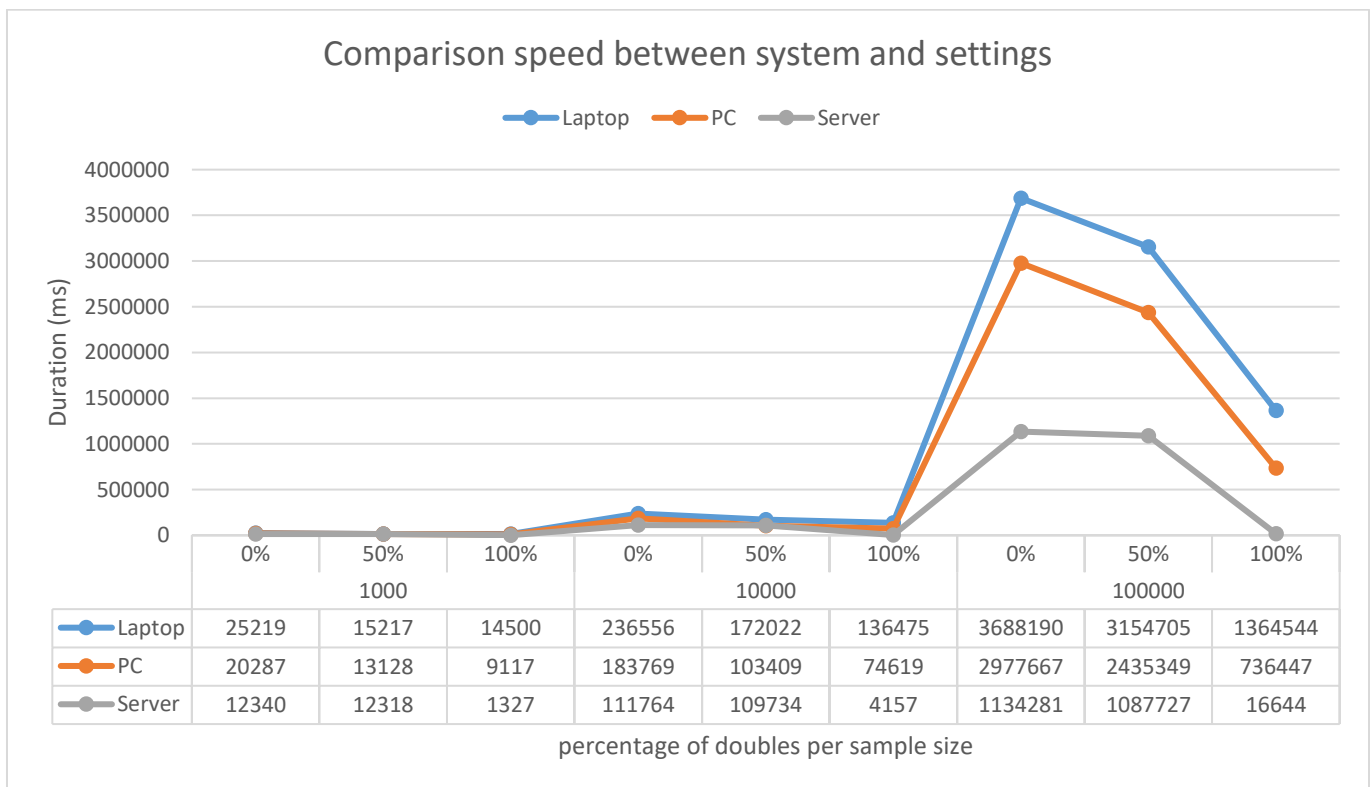


Figure 15 – Database module speed comparison graph

As can be seen in figure 15 above there are some wide variations of the performance between the different systems. A pattern can be found when comparing the results of both the laptop and pc tests. With an increasing speed in the operations of the system when increasing the number of duplicates in the test set. This of course can be explained by the fact that a lot less insert operations need to be performed, which decreases the overall duration needed for the application to finish all its operations. There are however some fluctuations as to how much improvement this results in. As can be seen the speed improvements between the different settings within the 1000 and 10000 sample sizes are present but marginal.

sample size	doubles %	Laptop	PC		Server		
		relative duration (base 0% per sample size)	relative duration (base 0% of sample size 1000)	relative duration (base 0% per sample size)	relative duration (base 0% of sample size 1000)	relative duration (base 0% per sample size)	relative duration (base 0% of sample size 1000)
1000	0%	1.00	1.00	1.00	1.00	1.00	1.00
	50%	0.60	0.60	0.65	0.65	1.00	1.00
	100%	0.57	0.57	0.45	0.45	0.11	0.11
10000	0%	1.00	9.38	1.00	9.06	1.00	9.06
	50%	0.73	6.82	0.56	5.10	0.98	8.89
	100%	0.58	5.41	0.41	3.68	0.04	0.34
100000	0%	1.00	146.25	1.00	146.77	1.00	91.92
	50%	0.86	125.09	0.82	120.04	0.96	88.15
	100%	0.37	54.11	0.25	36.30	0.01	1.35

Table 14 - Relative duration comparisons of database module speed test

Table 14 states the relative duration of the tests when compared within the sample size per system as well as when the sample size with 0% doubles is used as base for all instances. As can be seen, both the laptop and pc tests have a similar pattern with an increasing speed for the test sets containing only doubles when increasing the sample size and a decreasing relative speed when using 50% doubles. Another pattern which becomes apparent is that the increase of the 100% doubles instances increase approximately with multiplication of 10 which is the same increase as the sample size. The other two instances have this same pattern when looking at the lower sample sizes however this factor increases significantly when looking at the increase in relative duration from a sample size of 10000 to 100000. This is most probably explained by the fact that the number of insert operations that the system needs to perform increases a lot and this operation takes relatively a lot of time.

The server test results do not follow the same pattern as the results from the laptop and pc tests. They are a lot faster, which can easily be explained by the fact that all the select and insert operations are performed locally instead of via an internet connection. Another apparent variation is that of the immense speed increase when using only doubles, in comparison with the two other systems. Which, again, can be explained with the fact that these operations are now performed locally and thus take far less time. Lastly, one can see that the values between the test sets containing 0% and 50% duplicates are almost similar. When looking at the raw data in appendix F.2 – F.4 it becomes clear that on the server side some complications arose when performing the 50% duplicates containing tests, which resulted in almost as much insert operations as the tests containing 0% duplicates. This impacted the results immensely and is something we were not able to find an explanation for. We presume that it is a result of the difference in processing speed between select and insert statements. The reason for this assumption is that the problem was visible across all test sets, but worsened when performing on the local machine.

To conclude we see that our system performs as fast as can for this type of connection, most of the fluctuations in speed can be explained by differences in internet speed and connections. Also, the number of insert operations takes a heavy toll on the speed of the overall operations. Locally performed operations increased the speed of the application immensely however also resulted in faulty insert operations and errors.

6 Conclusion, Discussion and Future Implementations

6.1 Conclusion

In this chapter we will summarize and conclude our findings gathered during this project. We will address the research questions and sub questions as mentioned at the start of this paper.

“How can we find a method to correct location data in existing databases and provide an easy process for further data entries?”

To answer the main research question, we constructed a set of sub research questions to address the underlying issues to create a process for the location data correction and alteration.

“What is the fastest and most reliable way to add and alter the location data in the existing database, making this data complete and consistent?”

To create a process for the user to fast and reliably check location data we created a module consisting of a parent and multiple child processes. This way the parent process could set up the child processes, which would perform the actual execution of the location data alteration and check processes. Combining this set up with a pipelined process within the child processes increased the speed of the overall execution. The slowest set up combination as tested took a total of 246 seconds to perform a total of 2000 requests to Google, which is a vast improvement to performing this process manually. All this is set up by the user, by writing a simple configuration file to set up all the right information before the procedures, making this process as little time consuming as possible. To further increase the ease of use of the system, the API component of the system makes it possible to set up a series of tests sequentially and from external systems.

“What formatting and input guidelines should be used for future data entries to keep the location data of the entries added to the database complete and consistent?”

The system is set up to always convert the GPS location data to DD type of formatting. Therefore, the GPS location formatting used in the input data can either be DD or DMS type of formatting. To increase the effectiveness of the system it is however recommended to provide as much location data as possible and/or to use English notation for the location data when using the system. During testing it became apparent that the number of errors decreases when using either of the mentioned rules.

To conclude, it is clear that the method used to create a consistent and reliable, while also fast process to check and alter location data consists of a combination of techniques used. The combination of child processes, pipelining, configuration files, an API setup and the use of an external location data API resulted in an easy, reliable and fast process to check and alter GPS location data. The effectiveness of the system is, however, subject to the input as provided by the user.

“How can we provide a process to easily organize, transform, combine and check databases?”

To create a reliable process for the transformations of databases we created a setup of parent and child processes. This split system setup resulted in a streamlined process for the alterations of multiple databases and data input files simultaneously. The direct insert method as used within the system, creates the most reliable process and connection with the external databases, although this effected the speed of the process negatively. The combination of the created API process and use of configuration files created an easy approach for the user with a minimum amount of manual work

To conclude, it is clear that the method used to perform the process of database alterations created an easy and streamlined process for the user. The combination of the direct insert strategy, configuration files and the API module, created an easy process with minimal supervision needed by the user.

“How can we build a sustainable system that can add and verify GPS location data to databases while also being able to organize the data, and is compatible with different kind of databases?”

To answer the main research question, we constructed a set of sub research questions to address the underlying issues when creating a combined sustainable system.

“How can we design the system to operate both processes effortlessly alongside each other within one system?”

As can be seen in the chapters 3 and 4 we have created an overall module to fetch and handle the communication with the operating modules and the user. This module handles the startup operations and further command signaling of the application. The two operating modules are modularly and similarly designed, making use of parent and child processes. The upside to this is that if errors occur, the underlying reasons can easily be determined, making the system more reliable. The accompanying webserver module substantiates this effect and makes the communication of the user (or an external program) with the system easier.

“How can we design the system to be sustainable?”

A couple of factors contribute extensively towards the sustainability of the system:

1. The modular design of the system makes it possible for future alterations to be made without the need for rewriting major components of the system.
2. The possibility to run the main operating modules combined or separately makes it possible for a wide range of applications of the system.
3. The design of the major modules to make use of parent and child processes, increases the capacity of the modules. Increasing the speed and availability of the system.
4. The design is made to be easily expanded with extra functionalities, f.e. the possibility to easily expand the database component to be compatible with other database systems.

All the above-mentioned functionalities contribute towards the sustainability of the system.

To conclude a combination of modular design, split processes, configuration files and the support of the API server side makes the system run the two operational functionalities effortlessly alongside each other. Combine this with the possibility to easily alter and add functionalities to the designed system, to result in an overall sustainable and future-proof system.

6.2 Discussion

A couple of issues arose while designing and testing the built system. The major choices and issues are discussed in this section, explaining the made choices and ways to avoid problems when using the system.

6.2.1 Location module accuracy

When implementing the module for the checking of the GPS location data compared to given areas, an issue arose when this area had to be determined. To keep the complexity and thus speed of the system operationally feasible we decided to create a rectangle area of the 2 outmost points of the area. Figure 15 and 16 show the difference of these area's using the Netherlands as example.



Figure 16 - Map of the Netherlands



Figure 17 - Map of the Netherlands with borders used by system

As can be seen in the figures 16 and 17 above, this adds a lot of extra territory in neighboring countries as well as the North Sea. The percentage of added area differs widely between areas. To keep this to a minimum, we advise to use as detailed added location data as possible for the best results.

Our choice to implement this type of location checking influenced the accuracy of the application. By the decision to use the user-defined coordinates for the checking of the information, we were forced to make a trade off decision between speed and accuracy. In our opinion, in this way we could deliver both an adequate accuracy while maintaining an efficient process.

6.2.2 Google Geocoding limits

Another issue which arose during the testing of the location module was the limits as given within the Google Geocoding API (Google, Developer Guide, 2019). A couple of restrictions are given for the (free) use of the API:

1. A maximum of 40000 requests to make use of the API for free.
2. A maximum of 5000 requests per 100 seconds.
3. A maximum of 50 requests per second.

Because the system and module are both designed to be as efficient as possible, problems can arise when operating with a great number of sets simultaneously or when using sets of large sizes (1000 data entries and more). We therefore recommend to keep the testing restricted to smaller sets and split sets above the given threshold. Furthermore, we advise to run the system with one set at a time.

6.2.3 Spelling issue

The last issue of the location module is that the Google Geocoding API preferences English spelling and places in the USA. Therefore, when using other languages and or little area specifications, it can happen that the system wrongly alters GPS coordinates to places in the USA with a similar name. Please be aware, and use as detailed information as possible as well as omitting the use of abbreviations, when using the application.

Our choice to use the Google Geocoding API influences the results given as correct and incorrect, when improving coordinates. It also results in a domination of USA based locations and the English language and spelling. It could be possible that when implementing another API for the checking of coordinates, these results could differ from the current situation.

6.2.4 Database insert issue

During the testing of the Database module, when running both the application and database on the same server, the speed of the application increased significantly. However, this also increased the number of faulty insert operations. As this problem was also present when testing on the other systems, we suspect this to be the result of a difference in the speed at which the select and insert operations are being performed. Our decision to implement the direct insert operation instead of a cache first operation did influence these results, and with the implementation of a cache first system we would expect these results to improve.

6.3 Future implementations

A couple of implementations which could contribute to the functionality and ease of use of the system are mentioned in this section as well as possible future usage of the application.

6.3.1 UI

Currently to use the system, the user will have to use a terminal (and the possible API setup) to use the designed system. To further increase the ease of use of the system for the end user, a User Interface could be designed.

6.3.2 Cache First

Within the database module we chose for the direct insert type of connection when performing database operations. This creates a more secure, but also less fast connection with the connected database system. Another option would be to implement a cache first system, this would locally process all transformations to upload all at once after this process is finished. This database connection type would be more prone to errors and connection failing, however could increase the speed of the operations.

6.3.3 Reverse location lookup

The current lookup system works by using the additional location information to create a possible area the GPS coordinates need to be part of, to check for the correctness of the coordinates. Another way would be to perform a request using the given GPS coordinates and check for the received information with the added location information. Essentially to perform the operations in a backwards way of the process currently in use by the system. This could result in an increased accuracy of the system, especially when making use of both techniques combined.

6.3.4 GeoNames layer

As mentioned in Chapter 2 we chose to perform the process of checking and improving geographic coordinates using the Google Geocoding API because of the option to create areas of interest. A problem which arose was the fact that the API prefers English spelling and American cities. Although the GeoNames API would not be ideal for this process, because it only provides one set of coordinates instead of a bounding box, it could still be useful to implement it as an extra layer for the preprocessing of names. The API could be used to translate all location information into English as well as provide missing additional information about areas of interest, before we would check those locations against googles Geocoding API.

6.3.5 Dataset usage

In the near future the application will be used for the checking and improving of datasets containing geographic coordinates as well as additional location information. Examples of these datasets are a Fungi dataset analyzed within the Leiden University by Irene Martorelli and a dataset containing information regarding the living area of Butterflies found in the Indonesian Island Java. For both these datasets it is important to know exactly where these organisms are detected. Therefore, both the Fungi and Butterfly datasets can make use of the system to improve the consistency and integrity of the mentioned datasets. The application can further be used for any dataset in need for improvement of geographic coordinates, which can be performed direct inserting this information into the corresponding database as well as providing separate output files.

Bibliography

- About Node.js*. (n.d.). Retrieved from nodejs.org: <https://nodejs.org/en/about/>
- csv.js.org*. (n.d.). *CSV for Node.js*. Retrieved from csv.js.org: <https://csv.js.org/>
- DB-Engines. (2019, July). *DB-Engines Ranking*. Retrieved from db-engines.com: <https://db-engines.com/en/ranking>
- Decimal degrees*. (n.d.). Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Decimal_degrees
- DMS vs DD*. (2018, February). Retrieved from GISGeography: <https://gisgeography.com/decimal-degrees-dd-minutes-seconds-dms/>
- Geographic coordinate system*. (n.d.). Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Geographic_coordinate_system
- Google. (2019, June). *Developer Guide*. Retrieved from google.com: <https://developers.google.com/maps/documentation/geocoding/intro>
- Google. (n.d.). *Developer Guide*. Retrieved from developers.google.com: <https://developers.google.com/maps/documentation/geocoding/intro>
- Google Maps*. (n.d.). Retrieved from wikipedia.org: https://nl.wikipedia.org/wiki/Google_Maps
- Introducing JSON*. (n.d.). Retrieved from json.org: <https://www.json.org/>
- Javascript*. (n.d.). Retrieved from wikipedia.org: <https://en.wikipedia.org/wiki/JavaScript>
- Kommagescheiden bestand*. (n.d.). Retrieved from wikipedia.org: https://nl.wikipedia.org/wiki/Kommagescheiden_bestand
- MariaDB. (n.d.). *About MariaDB*. Retrieved from mariadb.org: <https://mariadb.org/about/>
- Ministerie van Binnenlandse Zaken. (2012). *Bouwbesluit Online 2012*. Retrieved from [rijksoverheid.bouwbesluit.com:](https://rijksoverheid.bouwbesluit.com/) https://rijksoverheid.bouwbesluit.com/Inhoud/docs/wet/bb2012_nvt/artikelsgewijs/hfd4/afd4-4/algemeen
- MonetDB.org. (n.d.). *Documentation*. Retrieved from monetdb.org: <https://www.monetdb.org/Documentation>
- MySQL-Enterprise. (n.d.). *MySQL Documentation*. Retrieved from dev.mysql.com: <https://dev.mysql.com/doc/>
- Node.js File System Module*. (n.d.). Retrieved from w3schools.com: https://www.w3schools.com/nodejs/nodejs_filesystem.asp
- Node.js. (n.d.). *Node.js v12.7.0 Documentation*. Retrieved from nodejs.org: <https://nodejs.org/api/http.html>
- NPM. (2016, August). *monetdb*. Retrieved from npmjs.com: <https://www.npmjs.com/package/monetdb>
- NPM. (2019, July). *mariadb*. Retrieved from npmjs.com: <https://www.npmjs.com/package/mariadb>

NPM. (2019, May). *mysql*. Retrieved from npmjs.com: <https://www.npmjs.com/package/mysql>

Web Services. (n.d.). Retrieved from Geoname.org: <https://www.geonames.org/export/web-services.html>

Appendix A – Readme

README.md

7/24/2019

Quick User Guide

Bachelorthesis Tim van Polen & Jelle sinnige (2019)

Here you will find quick reference information for using the program. For info about writing the configuration files, see the configuration syntax guide.

Setting up

Before running the application a few actions need to be performed, otherwise fatal crashes will occur.

- The most obvious first: Install NodeJS, this can be downloaded in a browser at <https://nodejs.org>, or installed via package manager.
- Install the necessary packages, using `npm install packageA packageB packageC ...`. The following packages are needed:
 - http
 - https
 - request
 - monetdb
 - mariadb
 - mysql2 (mysql has no prepare support)
 - csv
 - querystring
 - url
 - net
 - uuid
- **optional** Set the settings to your liking, in the files `server/database.ini` and `server/locations.ini` some basic settings for the program can be found, including which ports it needs to listen for, and which files are to be used for setting up HTTPS servers. The syntax for these files are as following: `setting = the value of the setting here, may contain spaces or be json objects`. The following settings can be set:
 - For the database module:
 - `port` the port to let the API server listen to
 - `db_default_config_object` the object to reference the given configuration to, **beware** change this only when you know what you're doing.
 - `https_cert` the HTTPS certificate to setup the https server with.
 - `https_key` the HTTPS key to setup the https server with
 - For the locations module:
 - `port` the port to let the API server listen to
 - `location_names` the default location names the program will extract from the target files, in case there are no valid names given. It is however not recommended to rely on this setting, instead, include places in the configuration JSON, see the [configuration syntax guide](#)

README.md

7/24/2019

- `coordinate_names` the same as `location_names`, but then for `latlng` coordinate columns
- `latitude_names` the same as `location_names`, but then for `lat` columns
- `longitude_names` the same as `location_names`, but then for `lng` columns
- `csv_delimiter` the default csv delimiter to use, will not be used by the helpers
- `https_cert` the HTTPS certificate to setup the HTTPS server with
- `https_key` the HTTPS key to setup the HTTPS server with
- `loc_default_config_object` the object to reference the given configuration to, **beware** change this only when you know what you're doing
- `google_api_key` the default Google Geocoding API key to use
- For the webserver module:
 - `port` the port the webserver will listen to
 - `https_cert` the HTTPS certificate to setup the HTTPS server with
 - `https_key` the HTTPS key to setup the HTTPS server with

Starting up

`node run` is used for starting the program, the following arguments can be given:

- `-verbose#` can be given to give more output. 0 is only errors, 1 errors and warnings, 2 errors warnings and information and 3 and 4 are for debugging. It is generally not a good idea to enable debugging for performance reasons.
- `-noWeb` can be given to disable the webserver module. It will not prevent the api's from starting up their endpoints.
- `-noLoc` can be given to disable the locations module.
- `-noDB` can be given to disable the database module.
- `-crashAlone` will keep the rest of the modules running if one of them crashes.
- `-noColor` will disable the color in the terminal.

Modules can be run as standalone, by navigating to `server` and running `node moduleNameHere`. They can have the following arguments:

- `-verbose#` see above
- `-cfgfile FileNameHere` for loading a custom configuration file. This is not a user configuration, as needed for processing files. It will only host the basic settings (like ports to run on).
- `-disableServer` disables the API endpoints.
- `-port` set API to run at a different port.
- `-noColor` disables the color in the terminal.

Runtime commands

These commands can be inserted in the console when the program is running.

In the run module:

- `exit` for exiting the program.
- `moduleName moduleCommand` by naming the module first and then naming the command to execute, for example: `database process test/operationCfgFile.json`

In the Database and Location module:

README.md

7/24/2019

- `process pathToFileToProcess` orders the database module to process a configuration file specifying what operations are to be performed. The syntax of such a file is found in `Configuration_Syntax_Guide.md`
- `mute #, #... N` Specify a list of helpers to mute, when one wants to ignore the output of certain helpers. These numbers correspond to the datasets in the configuration file; for each a helper will be created.
- `exit` for exiting the program

In the Webserver module:

- `exit` For exiting the process.

API Endpoints & Usage

Quick explanation: The APIs will listen to the ports defined in the ini files respectively. For quick reference, they are also mentioned by the terminal output upon program startup. They will present themselves in `INFO` messages, so they can be disabled by setting the `-verbose0` or `-verbose1` argument.

Reaching the API is as simple as making a `GET` or `POST` request to a certain address. The main address is always formatted as `https://localhost:600xx/endpoint/` where `600xx` is the port, and `endpoint` the endpoint that we want to reach. For simple lookups usually `GET` is required, and `POST` is used for doing more advanced or complex queries.

For the Locations Module:

The location module has the following endpoints:

- `/process` This endpoint is for starting a job on the locations module. For this endpoint a `POST` request is required, consisting of a single parameter: `request`. This parameter must contain a JSON object, in accordance with the [configuration syntax guide](#). This will return an object with 3 items:
 - `status` to show the status of the command, usually `success` or `error`
 - `msg` a message giving more detailed information if available
 - `key` a key to be used in the other endpoint.
- `/key/YOUR-KEY-HERE` This endpoint is for retrieving the status of a job, a process command initiated earlier. The key will be supplied to the issuer of the command, after the start. If the key is used, for example: `http://localhost:60012/key/abcde-fghijkl-mnopqrstuvw-xyz`, it will return an object containing the status of the job as a whole, and helpers individually. This needs to be checked periodically until the main status is either `crashed`, `failed` or `finished`.

For the Database Module:

The database module has the following endpoints:

- `/process` This endpoint is for starting a job on the database module. For this endpoint a `POST` request is required, consisting of a single parameter: `request`. This is a JSON object, in accordance with the [configuration syntax guide](#). This will return an object with 3 items:
 - `status` to show the status of the command, usually `success` or `error`
 - `msg` a message giving more detailed information if available
 - `key` a key to be used in the other endpoint

README.md

7/24/2019

- `/key/YOUR-KEY-HERE` This endpoint is for retrieving the status of a job, a process command initiated earlier. The key will be supplied to the issuer of the command after the start. If the key is used, for example `https://localhost:60011/key/abcde-fghijkl-mnopqrstuvw-xyz`, it will return an object containing the status of the job as a whole, and helpers individually. This needs to be checked periodically until the main status is either `crashed`, `failed` or `finished`.
- `/connect` is a special endpoint that also requires a POST request. However, this only requires basic database connection information, and will simply test if a connection to a certain database can be made. It requires the following parameters:
 - `host` the host of the external database
 - `user` the user of the external database
 - `password` the password for logging in into the external database as user
 - `database` the database to connect to
 - `port` the port to use
 - `type` the database type, can only be `mysql`, `monetdb` or `mariadb`
 - `ssl` **optional** a boolean to enable SSL
 - `sslcert` **optional** the contents of the certificate required to connect.

It will return a JSON object containing:

- `status` containing `success` if the connection succeeded, `failed` if it failed
- `msg`, containing the error message if the connection failed. It will be absent if the status is `success`

Appendix B – Configuration Syntax Guide

Configuration syntax guide

Bachelorthesis Tim van Polen & Jelle Sinnige (2019)

Below you will find a quick summary for the syntax that the modules expect in their datafiles.

A quick explanation

These files are basically JavaScript objects defining how a certain operation needs to be performed. It specifies what files are to be parsed and what files to output to. The format of these files is JSON, that means giving a file that is not JSON parsable will cause it to fail. The main body of these files consists of a `datasets` element, which is an array containing all the datasets to process. Each of these will be run in a separate process.

The Location Configuration Syntax

A configuration file the Locations module would accept looks like the following:

```
{
  "datasets": [
    {
      "source": String,
      "target": String,
      "lat": Array of Strings,
      "lng": Array of Strings,
      "latlng": Array of Strings,
      "places": Array of Strings,
      "margin": Number,
      "google_api_key": String,
      "includeOld": Boolean,
      "includePlaceId": Boolean,
      "includeCorrected": Boolean,
      "favorNames": Boolean,
      "delimiter": Character
    },
    {
      ... There can be more blocks, but there must be one at least ...
    }
  ],
  "path": String
}
```

Properties explained:

- `source` Source (input) File Relative to Command Path .csv/.json.
- `target` Target (output) File Relative to Command Path .csv/.json.
- `lat` An array of names, whose columns in the source file contain the Latitude values.
- `lng` An array of names, whose columns in the source file contain the Longitude values.

- **latlng** An array of names, whose columns in the source file contain the Latitude & Longitude in a string together. (Either this must be set or **lat** & **lng** must be set.)
- **places** contains a list of names whose columns in the source file contain name values, like country, areal code, placenames, streetnames. These columns must be given in descending order of location size, e.g. first continent, then country, then state, then city and finally address. Note that unavailable data can be omitted, using one column is enough but might not give the desired result, the more columns used, the more specific the searches will be.
- **margin** the margin of error the coordinates may have. 0.001 relates to ~ 1.1Km, 1.0 being exactly one degree (default set to 0.001)
- **google_api_key** the google api key to be used.
- **includeOld** *optional* if true, the output will print the old values as **Old_Latitude**, **Old_Longitude** and **Old_Name**, defaults to false if not set.
- **includePlaceId** *optional* if true, Google's place ID will be outputted as **PLACE_ID**, defaults to false if not set.
- **includeCorrected** *optional* if true, will print **Corrected_Lat** & **Corrected_Lng**, which are booleans, yes when the Latitude & Longitude have been altered by the system, no otherwise, defaults to false if not set.
- **favorNames** *optional* if true, the program will favor names above coordinates, meaning it will rather go to the coordinate that evaluates to the given name, if false or unset it will try to get the most out of the coordinate. It will attempt a less specific query to google, trying to fit the coordinates until it does. This method gets the most out of the coordinates. If true, more results may be corrected by the system, but you tend to end up with more specific names, if false, more coordinates will be preserved, but the names may be less specific than they were before, defaults to false if not set. Eg. **The Netherlands**, **Leiden** may be changed to **The Netherlands** if that makes the coordinates fit, however, the original coordinates are preserved. With **favorNames** the coordinates provided by Google for **The Netherlands**, **Leiden** will be outputted instead.
- **delimiter** *optional* if set, will override the default CSV delimiter, **,**. Only has effect when using CSV for input or output.
- **path** *only required when doing an API call* contains the path where the program has to start looking for the specified files. If not set, it will be set to the working directory of the program. If the process command given through the terminal, this parameter is not needed because the path will be derived from the path to the given file in the terminal.

The Database Configuration Syntax

A configuration file the Database module would accept looks like the following

```
{
  "datasets": [
    {
      "database": {
        "host": String,
        "port": Number,
        "user": String,
        "password": String,
        "database": String,
        "type": String,
```

```

    "ssl": {
      "ca": String
    }
  },
  "data": [
    {
      "table": String,
      "id": String,
      "method": {
        "insert": Boolean,
        "update": Boolean,
        "check_first": Boolean
      },
      "source": {
        "file": String,
        "type": String,
        "delimiter": String
      },
      "mapping": [
        {
          "source": String,
          "const": String,
          "target": String,
          "type": String,
          "important": Boolean,
          "null": Boolean,
          "transform": [
            {
              "type": String,
              "value": Any,
              "value2": Any
            },
            {
              ... Can be filled with 0 or more blocks ...
            }
          ],
          "where": [
            {
              "source": String,
              "target": String,
              "null": Boolean,
              "transform": [
                {
                  ... Can be filled with 0 or more blocks ...
                }
              ]
            },
            {
              ... There can be more blocks, but there must be one at least ...
            }
          ]
        },
        {
          ... There can be more blocks, but there must be one at least ...
        }
      ]
    }
  ]
}

```


Configuration_Syntax_Guide.md

8/23/2019

```

    }
  ]
},
{
  ... There can be more blocks, but there must be one at least ...
}
],
{
  ... There can be more blocks, but there must be one at least ...
}
],
"policy": String,
"unique": Boolean,
"integrity": String,
"path": String
}

```

Properties explained:

- **datasets** contains the datasets the database module has to upload.
 - **database** is an object containing the database connection properties
 - **host** the host to connect to
 - **port** the port to connect to
 - **user** the user to connect as
 - **password** the password to connect
 - **database** the database to use
 - **type** the type of database, can be **monetdb**, **mariadb** or **mysql**
 - **ssl** **optional** an object containing the path to the ssl certificate to connect with
 - **ca** the certificate file path, must be set if **ssl** is set
 - **data** is an object containing the data options, here is described what property will be uploaded to which database, table and column, and what transformations are to be performed upon the data before uploading.
 - **table** is the table this data is mapped to
 - **id** is the id of the table
 - **method** is an object containing instructions what to do with the dataset
 - **insert** set true if the data needs to be inserted
 - **update** set true if the data needs to be updated (*unsupported but expected!*)
 - **check_first** if to check if the record exists before inserting, or to just blindly insert
 - **source** is an object containing properties of the file that needs to be opened
 - **file** is the path to the file, relative to the configuration file
 - **type** is the type of the file, only JSON or CSV are supported
 - **delimiter** is the delimiter of the CSV file, only used when opening CSV files
 - **mapping** is an object describing how to map the data to the table. Each object in here represents a mapping of a column in the source file to a column in the target file.
 - **target** is the column to put the data in
 - **source** is the property from the source file to read, but it can also be a reference to another column in another table. By setting this as **database.table.column** the

program will search that column in that table for a certain value. In the case this is used the `where` property must be set! In regular columns, dots in the column name are not supported.

- `const optional` can be set in place of `source`, this will be treated as a constant, useful when the whole dataset needs a certain column to be filled with a certain value. It would be inefficient (however possible) to add this value to every record in the source file, because of the extra file reading and parsing overhead. **Watch out**, either `source` or `const` must be set, but never both in the same mapping object
- `type` the type of the value, can be `int`, `float`, `string`, `bool` etc.
- `important optional` if set, this value will be checked in the `WHERE` clause of an update statement. Does nothing for now.
- `null optional` if set, this value can be `null` and special queries will be generated. If the source value is `null` but this is not set, the record will be considered invalid and not set.
- `transform` is an array of objects describing how to transform the data before checking and uploading. This can be an empty array.
 - `type` the type of transformation to perform, can be any of the following:
 - `type` changes the type to the given type in `value`
 - `regex` matches the value to a given regular expression
 - `regex_first` returns the first match of a regular expression to the value
 - `modulo` performs a modulo on the source data
 - `divideby` performs a divide on the source data
 - `dividebyfloat` performs a divide on the source data, saves the answer as a float
 - `character` gets a certain character from a string
 - `substring` or `substr` returns a substring of a string
 - `value` always returns the same value (useful for constants)
 - `if` a condition, if the data from the source file equals the `value` property of the transform object, it will be set to the `value2` property
 - `add` to add a value on top of another value
 - `sub` to subtract a value from another value
 - `value` is the value of the operation above, by for example modulo, this is the amount to modulo by
 - `value2 optional` is a special value, only needed in some transformations where more than one value is needed, these are `if` and `substring`, however, if `substring` has no `value2` property, it will assume the length of the string
- `where optional` is an array of objects containing the conditions of the source, when it is an external value from another database/table. This must be omitted when the value is to be retrieved from the source file, as opposed to another database/table. To get the external value described in the `source` above, these values will be matched to the external database. It will create a SQL query: `SELECT the_external_value FROM the_external_table.the_external_database WHERE prop1 = value1... etc`
 - `source` the value to be extracted from the datafile. This time it **can't** reference another table, and must be a column in the source file.

Configuration_Syntax_Guide.md

8/23/2019

- `target` the column it is to be matched against in the database
- `null` **optional** if set, the value from the file can be `null` and a custom query will be generated. If this is `false` or unset, and the value from the file is `null`, the record will be deemed invalid and skipped
- `transform` an array with transformations to perform on the source data, can be empty
- `policy` the policy to abide by, at this moment, can only be set to `direct_insert`
- `integrity` **optional** does nothing for now, can be omitted

Appendix C – Usage example

System Usage Example

Bachelorthesis Tim van Polen & Jelle Sinnige (2019)

Below you will find an example of the usage of our components. This file will reference both the [Configuration_Syntax_Guide](#) and the [README](#) file.

Where to start

For our example project we have two files that need to be processed. A JSON file, `example.json`, and a CSV file, `example.csv`. There are two operations that we can perform on either file:

- Process the location data
- Upload it to an external database

We will describe both actions separately, but they can be combined.

Processing location data

Creating the configuration file

The common denominator in both our modules is JSON configuration files. A JSON configuration file needs to be created in order to specify where the program can find certain data in your JSON/CSV datafile. The JSON file will also specify how the data is processed, and where to put it. In the [Configuration_Syntax_Guide](#) is specified what properties are possible and required, however, we will discuss them here in a more informal setting, the referenced file is rather for a quick lookup, since we expect the user to get more used to them the more he/she uses the program.

We always start with a `datasets` property. This property is always the root property in our file. The only property that is allowed outside this root property, is the `path` property, but this property is only necessary when performing an API request. We will discuss how to initiate the program later. The `datasets` property is an array, consisting of objects. Each object is a task to perform, and thus by defining multiple objects multiple tasks will be performed. We will describe one such task. A 'task', or element in the `datasets` array, has its own required properties. The most important ones are `source`, `target`, `lat`, `lng`, `latlng`, `places` and `favorNames`.

- `source` is the source file to be opened and processed. This must include the path to the file, relative from the path to this configuration file, it is therefore recommended to place the configuration file in the same folder as the `source` and `target` file.
- `target` is the file where the program will output its data into, the file that will be created.
- `lat` & `lng` are the properties that specify where in the source file the latitude and longitude can be found. This may be omitted when the `latlng` property is given.
- `latlng` is the property where a combination of the latitude and longitude can be found.
- `places` are the columns that have the location names in them, such as Continent, Country, State, City, Area Code, Postal Code, Street, etc.

Say we have the following input files: `example.json` and `example.csv`. The first file is structured in JSON, and we have 6 components: `latitude`, `longitude`, `continent`, `country`, `city`, and `area`. The second file is

Usage_Example.md

8/23/2019

structured in CSV, and we have 5 components: `latlongitude`, `continent`, `country`, `city`, `area`. We would start out creating our configuration file (in the same folder), creating a `datasets` property, with two empty objects inside, as such:

```
{
  "datasets": [
    {
    },
    {
    }
  ]
}
```

Then, we would add our source and target files:

```
{
  "datasets": [
    {
      "source": "example.json",
      "target": "example_output.json"
    },
    {
      "source": "example.csv",
      "target": "example_output.csv"
    }
  ]
}
```

Now we would look what properties or column(s) in our source file contain the latitude and longitude properties. We find these are `latitude` and `longitude` in the JSON file, but we only find a `latlongitude` property in our CSV file. Let's put that in the configuration file as well. Note that depending on the structure of the input file, only a `latlng` property, or both a `lat` and a `lng` property need to be specified, in case of the csv file this is a `latlng` property because it has a column with both the latitude and the longitude in it.

```
{
  "datasets": [
    {
      "source": "example.json",
      "target": "example_output.json",
      "lat": "latitude",
      "lng": "longitude"
    },
    {
      "source": "example.csv",
      "target": "example_out.csv",

```

2 / 18

Usage_Example.md

8/23/2019

```

    "latlng": "latlongitude"
  }
]
}

```

Then, we'd search for the columns or properties that contain places. We put these in an array in the `places` property, in order. Our configuration file would then look as follows:

```

{
  "datasets": [
    {
      "source": "example.json",
      "target": "example_output.json",
      "lat": "latitude",
      "lng": "longitude",
      "places": [
        "continent",
        "country",
        "city",
        "area"
      ]
    },
    {
      "source": "example.csv",
      "target": "example_out.csv",
      "latlng": "latlongitude",
      "places": [
        "continent",
        "country",
        "city",
        "area"
      ]
    }
  ]
}

```

Now, we have a working configuration file. However, there is one more property that changes the way the program operates: `favorNames`. It is recommended to always include this property. Setting it true means the program will correct the coordinates immediately when they are found to be incorrect, while setting it false will let the program search if the coordinates are correct on a wider scope. For example, if the name is specified as "the Netherlands Leiden", and the coordinates do not match this name, the program will search for "the Netherlands", and check if the coordinates then fit. Setting `favorNames` true will make the program work in favor of the names, if false, it will make the program work in favor of the coordinates. In the end, if the coordinates are a full mismatch (no scope correction would solve it), the program will always correct the coordinates. In this example want to keep as much coordinate data as possible, and don't really care that much about the names, so we set it to `false`. Adding this would give the following configuration file:

Usage_Example.md

8/23/2019

```

{
  "datasets": [
    {
      "source": "example.json",
      "target": "example_output.json",
      "lat": "latitude",
      "lng": "longitude",
      "places": [
        "continent",
        "country",
        "city",
        "area"
      ],
      "favorNames": false
    },
    {
      "source": "example.csv",
      "target": "example_out.csv",
      "latlng": "latlongitude",
      "places": [
        "continent",
        "country",
        "city",
        "area"
      ],
      "favorNames": false
    }
  ]
}

```

There are more properties that can be added to enhance the results, but those will not be discussed here. Please refer to the [Configuration_Syntax_Guide](#). The content of the example configuration can be found in [example_config.json](#).

Running the configuration file

Now we have a proper configuration file, we can use it in the Location module. If the module is not yet started, now is the time. It can be run with the default way to setup the program, executing `node run` in the root folder, or by just starting the location module, which would require to change the directory to the `server` directory, and running `node locations`.

There are two methods to run the configuration file through the module, via a command or via an API call.

Executing the configuration file as a command

Executing the configuration as a command is quite simple. If you ran the program as a whole, you'd first need to specify what module you want to command, then what it'd needs to do, and finally the file. A more detailed explanation of commands can be found in [Readme](#).

Usage_Example.md

8/23/2019

An example of the command to run would be: `locations process test/example/example_config_loc.json`.

If only the locations module is run, it would be: `process ../test/example/example_config_loc.json`. Note that when the locations module is run separately, `locations` must be omitted, because there would not be ambiguity between the modules, and the path starts with `../` because the module is run from within the `server` folder, as opposed to the `root` folder.

Executing the configuration file as an API call

Executing the command via API call is a little more complicated, and would require knowledge of programming (to some extent). You'd need a small piece of code that can make a simple web request to a webserver. When using JavaScript, one could use the `request` library for this.

Before sending the actual request, check your configuration file first. There is a special `path` property that needs to contain the relative path to the files in the configuration file. In our previous configuration file, this would look like this:

```
{
  "datasets": [
    {
      "source": "example.json",
      "target": "example_output.json",
      "lat": "latitude",
      "lng": "longitude",
      "places": [
        "continent",
        "country",
        "city",
        "area"
      ],
      "favorNames": false
    },
    {
      "source": "example.csv",
      "target": "example_out.csv",
      "latlng": "latlongitude",
      "places": [
        "continent",
        "country",
        "city",
        "area"
      ],
      "favorNames": false
    }
  ],
  "path": "test/example/"
}
```


Usage_Example.md

8/23/2019

First, you need to format your request properly. This is quite simple: there is only one `POST` parameter, namely `request`. `request` must contain the full configuration file as stringified JSON, the API will parse it.

The request must then be done to the `process` API endpoint of the locations module. The address for this endpoint is by default `https://localhost:60012/process`, but it can differ when it is not run locally, or setup to use another port. For more information about the API endpoints, please refer to the API section of the [readme](#) file.

The request will give a response with these 3 items:

- `status`: if an error occurred or not
- `msg`: a message accompanying this error or success status with a more detailed explanation.
- `key`: a key that is required to look up the status of the request later.

If a key was returned, your request has been processed successfully. Now we're about halfway done.

To check the status of the key, we'd need to make another API request to the locations module. This request is a little more simple, as it would only require a `GET` request, so there are no parameters. The request must be made to the `key` endpoint of the API, with the key then attached, so for example the full address would be `https://localhost:60012/key/YOUR-KEY-GOES-HERE`.

This API call would return an object with status elements of the whole job, and of the sub-processes individually. The status will be one of 4:

- `not finished` to indicate the process is still busy
- `finished` to indicate the process has terminated successfully
- `crashed` to indicate the program has crashed during runtime
- `failed` to indicate the process could not finish successfully.

If the status is `not finished` it is best to periodically check again until it is.

Processing database data

Creating the configuration file

For the creation of the configuration file we would go about the same way as the locations configuration file. First we'd look what input we have: a file named `example.csv` with 5 columns:

- `latlongitude`
- `continent`
- `country`
- `city`
- `area`

We start by creating an empty datasets array in our JSON configuration file:

```
{
  "datasets": [
```


Usage_Example.md

8/23/2019

```
  ]
}
```

After this, we want to set some important properties. These must be set in the root object:

- `policy`
- `unique`
- `integrity`

`policy` defines what method is to be used while uploading. The only supported method currently is `direct insert`. `unique` defines if the entry should be checked against the database before importing it. However, support for anything but `true` is not guaranteed. `integrity` was implemented for later implementation of other features, but as of now it should be set to `true`. We'd end up with the following configuration file:

```
{
  "datasets": [

  ],
  "policy": "direct_insert",
  "unique": true,
  "integrity": "full"
}
```

Then, we would create the datasets according to our input files. This is only one file, so one dataset would be enough. If you have multiple files, but they are to be uploaded into the same database, you also don't need to create multiple datasets objects.

```
{
  "datasets": [
    {

    }
  ],
  "policy": "direct_insert",
  "unique": true,
  "integrity": "full"
}
```

A dataset element should always contain the following two properties:

- `database`, where all the database connection properties will be described
- `data`, an array containing objects describing which data should be uploaded to this database. If there are multiple datasets that use the same database, it would be easier to put multiple entries in the `data` property, than to put multiple items in the `datasets` property, since every `dataset` element needs a `database` element with connection properties. We would now have this:

Usage_Example.md

8/23/2019

```
{
  "datasets": [
    {
      "database": {

      },
      "data": [

      ]
    }
  ],
  "policy": "direct_insert",
  "unique": true,
  "integrity": "full"
}
```

Next, we'd fill in the database connection properties. These are custom to what you have setup. They include:

- **host** what host the module has to connect to
- **port** the port
- **user** the user to connect with
- **password** the password to use
- **database** the database to connect to
- **type** the database type on the other end of the connection, MySQL, MonetDB or MariaDB are supported as of now.
- **ssl** an object containing:
 - **ca** the certificate to use for a encrypted connection.

We will not use an encrypted connection in this example. If we would fill in what we know, we'd get something like this:

```
{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [

      ]
    }
  ],
  "policy": "direct_insert",
  "unique": true,
}
```

8 / 18

Usage_Example.md

8/23/2019

```
"integrity": "full"
}
```

We now have a configuration that can connect, but does not do anything yet. For each file that we want to process to this database we'd create an object in the `data` array. In this case it would be one item. Multiple `data` elements can be made for the same file, if you want to split it up.

```
{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [
        {
          // ...
        }
      ]
    }
  ],
  "policy": "direct_insert",
  "unique": true,
  "integrity": "full"
}
```

This data element has some important properties:

- `table` the table to insert the items into
- `id` the id of that table
- `method` an object
- `source` an object
- `mapping` an array Let's fill these in into our `data` object:

```
{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      // ...
    }
  ],
  // ...
}
```

9 / 18

Usage_Example.md

8/23/2019

```

    "data": [
      {
        "table": "locations",
        "id": "loc_id",
        "method": {

        },
        "source": {

        },
        "mapping": [

        ]
      }
    ],
    "policy": "direct_insert",
    "unique": true,
    "integrity": "full"
  }
}

```

Now let's fill the `method` object: it defines the method to handle the database. It has three properties which are booleans:

- `insert` if we want to insert records of this dataset
- `update` if we want to update records of this dataset
- `check_first` if we want to check beforehand if the entries already exists. These booleans are mainly created for future implementations, it is best to set them to `true,false,true`. Let's fill this in:

```

{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [
        {
          "table": "locations",
          "id": "loc_id",
          "method": {
            "insert": true,
            "update": false,
            "check_first": true
          },
          "source": {

```

10 / 18

Usage_Example.md

8/23/2019

```

        },
        "mapping": [
            ]
        }
    ]
},
"policy": "direct_insert",
"unique": true,
"integrity": "full"
}

```

Now we want to fill the source object. It has also three properties, `file`, `type` and `delimiter`. `file` is the path to the input file, relative to the configuration file. `type` is the filetype and `delimiter` is the delimiter that is used in csv files. In this version even when opening JSON files a delimiter property is still expected, even when it is not used. We fill these in to our needs:

```

{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [
        {
          "table": "locations",
          "id": "loc_id",
          "method": {
            "insert": true,
            "update": false,
            "check_first": true
          },
          "source": {
            "file": "example.csv",
            "type": "csv",
            "delimiter": ","
          },
          "mapping": [
            ]
        }
      ]
    }
  ]
}

```

11 / 18

Usage_Example.md

8/23/2019

```

    "policy": "direct_insert",
    "unique": true,
    "integrity": "full"
  }

```

Finally, we come to the mapping array. This is an array of name pairs that define which column is uploaded into which column in the database. Each element in this array has at least the following properties:

- **source** the source column, which column or property in the file we want to specify
- **target** the column to insert it into
- **type** the type of the value, a string that can be "int", "date", "float", "string", "bool"
- **null** if the element can be null, this is a boolean
- **transform** an array with transformations.

Because we want to upload all 5 of our values from the source file, we create 5 objects inside the mapping array. Let's create 5 empty objects first:

```

{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [
        {
          "table": "locations",
          "id": "loc_id",
          "method": {
            "insert": true,
            "update": false,
            "check_first": true
          },
          "source": {
            "file": "example.csv",
            "type": "csv",
            "delimiter": ","
          },
          "mapping": [
            {
              },
              {
                },
                {
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }

```

12 / 18

Usage_Example.md

8/23/2019

```

        },
        {
            },
        {
            }
        ]
    }
}
],
"policy": "direct_insert",
"unique": true,
"integrity": "full"
}

```

Then we proceed to fill them with the required properties:

```

{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [
        {
          "table": "locations",
          "id": "loc_id",
          "method": {
            "insert": true,
            "update": false,
            "check_first": true
          },
          "source": {
            "file": "example.csv",
            "type": "csv",
            "delimiter": ","
          },
          "mapping": [
            {
              "source": "latlongitude",
              "target": "latlongitude",
              "type": "string",
              "null": true,
              "transform": [ ]
            }
          ]
        }
      ]
    }
  ]
}

```

13 / 18

Usage_Example.md

8/23/2019

```

    {
      "source": "continent",
      "target": "continent",
      "type": "string",
      "null": true,
      "transform": [ ]
    },
    {
      "source": "country",
      "target": "country",
      "type": "string",
      "null": true,
      "transform": [ ]
    },
    {
      "source": "city",
      "target": "city",
      "type": "string",
      "null": true,
      "transform": [ ]
    },
    {
      "source": "area",
      "target": "area",
      "type": "string",
      "null": true,
      "transform": [ ]
    }
  ]
}
],
"policy": "direct_insert",
"unique": true,
"integrity": "full"
}

```

If we'd want to specify an external key, we would in the `source` element of a `mapping` object specify the database, table and column separated by dots. So if we'd want column C from table B from database A, we'd say the source is `A.B.C`. In that case, also a `where` element should be added to the `mapping` object, to specify which properties should be set when looking up this value. Say we'd want an id of an user in another table. We say the source would be `"Database.usertable.id"`. We want that the name must be the same as in the `name` column in our file, and the email must be the same as our `email` column in our file. We'd then create a `where` property like this:

```

{
  "mapping": [
    {
      "source": "Database.usertable.id",
      "target": "user_id",

```

14 / 18

Usage_Example.md

8/23/2019

```

    "type": "int",
    "null": false,
    "transform": [ ],
    "where": [
      {
        "source": "name",
        "target": "name"
      },
      {
        "source": "email",
        "target": "email"
      }
    ]
  }
]
}

```

The transform array contains the transformations. In our example it is empty, since we don't perform any transformations. However, a sequence of transformations would look like this. Say we have a Fahrenheit value in our datafile, and a Celcius column in our table, so we want to subtract 32 first, then multiply it by 5 and divide it by 9. Our transform array would look like this:

```

{
  "transform": [
    {
      "type": "sub",
      "value": 32
    },
    {
      "type": "mult",
      "value": 5
    },
    {
      "type": "dividebyfloat",
      "value": 9
    }
  ]
}

```

Note the `dividebyfloat`, there is also a `divideby` operation, but it will save the value as an integer.

More details can be found in the [Configuration_Syntax_Guide](#).

Running the configuration file

Now we have a proper configuration file, we can use it in the Database module. If the module is not yet started, now is the time. It can be run with the default way to setup the program, executing `node run` in the root folder, or by just starting the database module, which would require to change the directory to the `server` directory, and running `node database`.

Usage_Example.md

8/23/2019

There are two methods to run the configuration file through the module, via a command or via an API call.

Executing the configuration file as a command

Executing the configuration as a command is quite simple. If you ran the program as a whole, you'd first need to specify what module you want to command, then what it'd needs to do, and finally the file. A more detailed explanation of commands can be found in [Readme](#).

An example of the command to run would be: `database process test/example/example_config_db.json`.

If only the database module is run, it would be: `process ../test/example/example_config_db.json`. Note that when the database module is run separately, `database` must be omitted, because there would not be ambiguity between the modules, and the path starts with `../` because the module is run from within the `server` folder, as opposed to the `root` folder.

Executing the configuration file as an API call

Executing the command via API call is a little more complicated, and would require knowledge of programming (to some extent). You'd need a small piece of code that can make a simple web request to a webserver. When using JavaScript, one could use the `request` library for this.

Before sending the actual request, check your configuration file first. There is a special `path` property that needs to contain the relative path to the files in the configuration file. In our previous configuration file, this would look like this:

```
{
  "datasets": [
    {
      "database": {
        "host": "jsit.nl",
        "port": 3306,
        "user": "bp_test",
        "password": "Q68auv1enmRma2y6!",
        "database": "bp_test",
        "type": "mysql"
      },
      "data": [
        {
          "table": "locations",
          "id": "loc_id",
          "method": {
            "insert": true,
            "update": false,
            "check_first": true
          },
          "source": {
            "file": "example.csv",
            "type": "csv",
            "delimiter": ","
          }
        }
      ]
    }
  ]
}
```

16 / 18

Usage_Example.md

8/23/2019

```

    "mapping": [
      {
        "source": "latlongitude",
        "target": "latlongitude",
        "type": "string",
        "null": true,
        "transform": [ ]
      },
      {
        "source": "continent",
        "target": "continent",
        "type": "string",
        "null": true,
        "transform": [ ]
      },
      {
        "source": "country",
        "target": "country",
        "type": "string",
        "null": true,
        "transform": [ ]
      },
      {
        "source": "city",
        "target": "city",
        "type": "string",
        "null": true,
        "transform": [ ]
      },
      {
        "source": "area",
        "target": "area",
        "type": "string",
        "null": true,
        "transform": [ ]
      }
    ]
  },
  "policy": "direct_insert",
  "unique": true,
  "integrity": "full",
  "path": "test/example/"
}

```

First, you need to format your request properly. This is quite simple: there is only one **POST** parameter, namely **request**. **request** must contain the full configuration file as stringified JSON, the API will parse it.

The request must then be done to the **process** API endpoint of the database module. The address for this endpoint is by default <https://localhost:60011/process>, but it can differ when it is not run locally, or

Usage_Example.md

8/23/2019

setup to use another port. For more information about the API endpoints, please refer to the API section of the [readme](#) file.

The request will give a response with these 3 items:

- **status**: if an error occurred or not
- **msg**: a message accompanying this error or success status with a more detailed explanation.
- **key**: a key that is required to look up the status of the request later.

If a key was returned, your request has been processed successfully. Now we're about halfway done.

To check the status of the key, we'd need to make another API request to the database module. This request is a little more simple, as it would only require a **GET** request, so there are no parameters. The request must be made to the **key** endpoint of the API, with the key then attached, so for example the full address would be <https://localhost:60011/key/YOUR-KEY-GOES-HERE>.

This API call would return an object with status elements of the whole job, and of the sub-processes individually. The status will be one of 4:

- **not finished** to indicate the process is still busy
- **finished** to indicate the process has terminated successfully
- **crashed** to indicate the program has crashed during runtime
- **failed** to indicate the process could not finish successfully.

If the status is **not finished** it is best to periodically check again until it is.

Appendix D - Support modules validation tests

D.1 Interface validation test code

```
//////interfaces test, Bachelorthesis 2019, Tim van Polen & Jelle Sinnige
'use strict'
```

```
let ifs = require('../server/interfaces.js');

ifs.setLogType(5);
ifs.setProgramTag("TestProgram");

console.log("Show all");

console.log("Show ERR message: ");
ifs.log_e("This is an error");
console.log("Show WARN message: ");
ifs.log_w("This is a warning");
console.log("Show INFO message");
ifs.log_i("This is an info message");
console.log("Show DEBUG message");
ifs.log_d("This is a DEBUG message");
console.log("Show data dump");
ifs.log_dd(['data', 'data', 'data'], "datadump");

console.log("Below there should not be any DEBUG or data dump:");

ifs.setLogType(2);

console.log("Show ERR message: ");
ifs.log_e("This is an error");
console.log("Show WARN message: ");
ifs.log_w("This is a warning");
console.log("Show INFO message");
ifs.log_i("This is an info message");
console.log("Show DEBUG message");
ifs.log_d("This is a DEBUG message");
console.log("Show data dump");
ifs.log_dd(['data', 'data', 'data'], "datadump");

console.log("Below there should only be an error");

ifs.setLogType(0);

console.log("Show ERR message: ");
ifs.log_e("This is an error");
console.log("Show WARN message: ");
ifs.log_w("This is a warning");
console.log("Show INFO message");
```

```

ifs.log_i("This is an info message");
console.log("Show DEBUG message");
ifs.log_d("This is a DEBUG message");
console.log("Show data dump");
ifs.log_dd(['data', 'data', 'data'], "datadump");

console.log("Show all messages without color.");

ifs.setDisableColor(true);
ifs.setLogType(5);

console.log("Show ERR message: ");
ifs.log_e("This is an error");
console.log("Show WARN message: ");
ifs.log_w("This is a warning");
console.log("Show INFO message");
ifs.log_i("This is an info message");
console.log("Show DEBUG message");
ifs.log_d("This is a DEBUG message");
console.log("Show data dump");
ifs.log_dd(['data', 'data', 'data'], "datadump");

```

D.2 Typehandler validation test code

```

//typehandler test, Bachelorthesis 2019, Tim van Polen & Jelle Sinnige
'use strict';

var typehandler = require('../server/typehandler.js');

console.log("Checking similarities: ");

let object_a = {
  prop1: 1,
  prop2: 4,
  prop5: "str"
};

let object_b = {
  prop1: 4,
  prop2: 10
};

console.log("Expect output true: ");
console.log(typehandler.check_similar_object(object_a, object_b));

let object_c = {
  prop2: "henkie",
  prop10: "7"
};

```

```

let object_d = {
  prop1: 124,
  prop2: "piet",
  prop10: "5"
}

console.log("Expect output false (failed on key count):");
console.log(typehandler.check_similar_object(object_c,object_d));

let object_e = {
  prop1: 100,
  prop2: 99,
  prop4: 97,
  prop5: 96
};

let object_f = {
  prop1: 100,
  prop2: 99,
  prop3: 98,
  prop4: 9
};

console.log("Expect output false (failed on not having required keys):");
console.log(typehandler.check_similar_object(object_e,object_f));

let object_g = {
  prop1: 200,
  prop2: 201,
  prop3: 202,
  prop4: 203,
  prop5: 204
};

let object_h = {
  prop1: 200,
  prop2: 201,
  prop3: 202
};

console.log("Expect output true (original may have more properties than
reference):");
console.log(typehandler.check_similar_object(object_g,object_h));

console.log("Checking copy and equal");
let obj = { a: "nee", b: "ja" };

console.log("Copying "+JSON.stringify(obj));

```

```

let copy = typehandler.create_copy_of_object(obj);
console.log("Expect this obj to be the same: "+JSON.stringify(copy));

console.log("Expect compare to be true");
console.log(typehandler.check_equal_object(obj,copy));

```

D.3 JSON stream and stringify validation test code

```

//JSONstream test, Bachelorthesis 2019, Tim van Polen & Jelle Sinnige
'use strict';

var { JSONstream } = require('../server/JSONstream.js');
var ifs = require('../server/interfaces.js');
var fs = require('fs');
var { stringifyJSON } = require('../server/JSONstringify.js');

ifs.setLogType(4);
ifs.log_d("Starting test.");

var jsonstr = JSONstream({});
var json_to_str = stringifyJSON();
var fstr = fs.createWriteStream('json_stream_output.json');
var rstr = fs.createReadStream('testjsonstream.json');
//piping to stdout wont work, stdout wont accept objects
rstr.pipe(jsonstr).pipe(json_to_str).pipe(fstr); //this works perfectly

```

D.4 DMS to DD transformation test code

```

//DMS to DD test, Bachelorthesis 2019, Tim van Polen & Jelle Sinnige
'use strict';

let coords = require('../server/coordinates.js');

let coord_1 = "52°14'36\",5°38'3\"";
let coord_2 = "-52°14'36\"S,5°38'3\"";
let coord_3 = "52°14'36\",-5°38'3\"W";
let coord_4 = "-52°14'36\"S,-5°38'3\"W";
let coord_5 = "52°14'36\"N,5°38'3\"E";
let coord_6 = "-5°38'3\"W,52°14'36\"N";
let coord_7 = "5°38'3\"E,-52°14'36\"S";
let coord_8 = "52.24333333,5.63416667";

console.log("coord_1 (" + coord_1 + ") evaluates to:
"+coords.convertSingleDMStoDD(coord_1));
if (coords.error) {
    console.log(coords.error);
}
console.log("coord_2 (" + coord_2 + ") evaluates to:
"+coords.convertSingleDMStoDD(coord_2));

```



```

if (coords.error) {
  console.log(coords.error);
}
console.log("coord_3 (" + coord_3 + ") evaluates to:
" + coords.convertSingleDMStoDD(coord_3));
if (coords.error) {
  console.log(coords.error);
}
console.log("coord_4 (" + coord_4 + ") evaluates to:
" + coords.convertSingleDMStoDD(coord_4));
if (coords.error) {
  console.log(coords.error);
}
console.log("coord_5 (" + coord_5 + ") evaluates to:
" + coords.convertSingleDMStoDD(coord_5));
if (coords.error) {
  console.log(coords.error);
}
console.log("coord_6 (" + coord_6 + ") evaluates to:
" + coords.convertSingleDMStoDD(coord_6));
if (coords.error) {
  console.log(coords.error);
}
console.log("coord_7 (" + coord_7 + ") evaluates to:
" + coords.convertSingleDMStoDD(coord_7));
if (coords.error) {
  console.log(coords.error);
}
console.log("coord_8 (" + coord_8 + ") evaluates to:
" + coords.convertSingleDMStoDD(coord_8));
if (coords.error) {
  console.log(coords.error);
}

//let coord_2 = "52.24333333,5.63416667";

```

D.5 Database SQL generation validation test code

//mysqlgen test, Bachelorthesis 2019, Tim van Polen & Jelle Sinnige

```

let sql = require('../server/databasesqlgen.js');

console.log("Expected: SELECT id FROM testtable WHERE ( item1 = ? AND item2 =
? AND item3 = ? )");
console.log(sql.query_select_simple("testtable", ["id"], ["item1", "item2", "item3
"], "database"));

```

```

console.log("Expected: SELECT * FROM testtable WHERE ( item1 = ? AND item2 = ?
AND item3 = ? )");
console.log(sql.query_select_all_simple("testtable",["item1","item2","item3"])
);

console.log("Expected: INSERT INTO testtable ( item1, item2, item3 ) VALUES (
?, ?, ? )");
console.log(sql.query_insert_simple("testtable",["item1","item2","item3"]));

```

D.6 Transform validation module code

```

//mysqlgen test, Bachelorthesis 2019, Tim van Polen & Jelle Sinnige
'use strict';

```

```

let tf = require('../server/transform.js');

```

```

//type - int - str - float - bool
//regex
//regex-first
//modulo
//div
//div-float
//char
//substr
//value
//if

```

```

console.log("transform types initial value float (1.2), expect string, number
(whole), number (comma), boolean")
let val = 1.2;
let t1 = [{ type: "type", value: "string" }];
let t2 = [{ type: "type", value: "int" }];
let t3 = [{ type: "type", value: "float" }];
let t4 = [{ type: "type", value: "bool" }];
let a1 = tf.transformSource(val,t1);
let a2 = tf.transformSource(val,t2);
let a3 = tf.transformSource(val,t3);
let a4 = tf.transformSource(val,t4);
console.log("a1: "+a1+" - "+(typeof a1));
console.log("a2: "+a2+" - "+(typeof a2));
console.log("a3: "+a3+" - "+(typeof a3));
console.log("a4: "+a4+" - "+(typeof a4));

```

```

console.log("transform regexes, input string abababa, regex = a");
let val2 = "abababa";
let t5 = [{ type: "regex", value: "a" }];
let t6 = [{ type: "regex_first", value: "a" }];
let t77 = [{ type: "regex", value: "c" }];
let a5 = tf.transformSource(val2,t5);

```

```

let a6 = tf.transformSource(val2,t6);
let a77 = tf.transformSource(val2,t77);
console.log("a5: "+a5);
console.log("a6: "+a6);
console.log("a77: "+a77);

console.log("transform numbers, modulo div and div-float, input is 10, other
input 4");
console.log("Expect 0 ( 20%4), 5 (20/4)  and 0.5 (20/40)");
let val3 = 20;
let t7 = [{ type: "modulo", value: 4 }];
let t8 = [{ type: "divideby", value: 4 }];
let t9 = [{ type: "dividebyfloat", value: 40 }];
let a7 = tf.transformSource(val3,t7);
let a8 = tf.transformSource(val3,t8);
let a9 = tf.transformSource(val3,t9);
console.log("a7: "+a7);
console.log("a8: "+a8);
console.log("a9: "+a9);

console.log("transform char and substring tests: string abcd");
let val4 = "abcd";
let t10 = [{ type: "char", value: 2 }];
let t11 = [{ type: "substring", value: 2 }];
let a10 = tf.transformSource(val4,t10);
let a11 = tf.transformSource(val4,t11);
console.log("a10: "+a10);
console.log("a11: "+a11);

console.log("transform condition and value test");
let val5 = "a";
let t12 = [{ type: "if", value: "a", value2: "b" }];
let t13 = [{ type: "value", value: "c"}];
let a12 = tf.transformSource(val5,t12);
let a13 = tf.transformSource(val5,t13);
console.log("expect b and c");
console.log(a12);
console.log(a13);

```

Appendix E - Location module tests

E.1 Specifications of the systems

LAPTOP	specs	Processor name	Intel Core i7 7700HQ
		CPU max speed	2.80 GHz
		CPU L3 cache size	6 MB
		CPU number of cores	4
		CPU number of threads	8
		Ram type	DDR4
		RAM size	2 x 8 GB
		RAM speed	2400 MHz
		Storage type	HDD
		Network speed	44,17 Mbps/ 19,91 Mbps (Download / Upload)
		Network ping	12 ms
		Network type	WIFI (2.4 GHz)
PC	specs	Processor name	Intel Core i7 4770K
		CPU max speed	3.90 GHz
		CPU L3 cache size	8 MB
		CPU number of cores	4
		CPU number of threads	8
		Ram type	DDR3
		RAM size	2 x 8 GB
		RAM speed	1333 MHz
		Storage type	SSD
		Network speed	92,89 Mbps/ 87,18 Mbps (Download / Upload)
		Network ping	9 ms
		Network type	LAN
Server	specs	Processor name	Intel Xeon E3-1240L v5
		CPU max speed	3.20 GHz
		CPU L3 cache size	8 MB
		CPU number of cores	4
		CPU number of threads	8
		Ram type	DDR4
		RAM size	2 x 8 GB
		RAM speed	2133MHz
		Storage type	SSD
		Network speed	100,00 Mbps/ 100,00 Mbps (Download / Upload)
		Network ping	3 ms
		Network type	LAN (local)

Table 15 - System Specifications

E.2 Laptop FavorNames True test results

Favor names aan

		run 1						run 2					
sample size	% correct	speed (ms)	failed	validated	corrected	#requests		speed (ms)	failed	validated	corrected	#requests	
10	0	1496	0	0	10	10		1548	0	0	10	10	
	50	1464	0	5	5	10		1481	0	5	5	10	
	100	1526	0	10	0	10		1608	0	10	0	10	
100	0	14856	0	0	100	100		13194	0	0	100	100	
	50	10965	0	50	50	100		13183	0	50	50	100	
	100	10948	0	100	0	100		14451	0	100	0	100	
1000	0	120895	0	0	1000	1000		127311	0	0	1000	1000	
	50	129242	0	50	50	1000		126491	0	50	50	1000	
	100	135751	0	1000	0	1000		128668	0	1000	0	1000	
		run 3						run 4					
sample size	% correct	speed (ms)	failed	validated	corrected	#requests		speed (ms)	failed	validated	corrected	#requests	
10	0	1283	0	0	10	10		1427	0	0	10	10	
	50	1187	0	5	5	10		1383	0	5	5	10	
	100	1255	0	10	0	10		1382	0	10	0	10	
100	0	12783	0	0	100	100		12729	0	0	100	100	
	50	11448	0	50	50	100		14019	0	50	50	100	
	100	11551	0	100	0	100		12677	0	100	0	100	
1000	0	121501	0	0	1000	1000		122460	0	0	1000	1000	
	50	119895	0	50	50	1000		124035	0	50	50	1000	
	100	112458	0	1000	0	1000		124904	0	1000	0	1000	
		run 5											
sample size	% correct	speed (ms)	failed	validated	corrected	#requests							
10	0	1239	0	0	10	10							
	50	1373	0	5	5	10							
	100	1233	0	10	0	10							
100	0	10424	0	0	100	100							
	50	10385	0	50	50	100							
	100	10835	0	100	0	100							
1000	0	105015	0	0	1000	1000							
	50	110855	0	50	50	1000							
	100	124167	0	1000	0	1000							

Figure 18 - Test results location speed test (FavorNames True, Laptop)

E.3 Laptop FavorNames False test results

Favor names uit

		run 1					run 2				
sample size % correct		speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	2837	0	10	0	20	2664	0	10	0	20
	50	2171	0	10	0	15	2400	0	10	0	15
	100	1405	0	10	0	10	1502	0	10	0	10
100	0	22634	0	100	0	200	26439	0	100	0	200
	50	17147	0	100	0	150	18958	0	100	0	150
	100	11420	0	100	0	100	12759	0	100	0	100
1000	0	261442	0	1000	0	2000	253295	0	1000	0	2000
	50	197102	0	1000	0	1500	190692	0	1000	0	1500
	100	131133	0	1000	0	1000	126810	0	1000	0	1000
		run 3					run 4				
sample size % correct		speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	2286	0	10	0	20	2577	0	10	0	20
	50	1829	0	10	0	15	1968	0	10	0	15
	100	1234	0	10	0	10	1373	0	10	0	10
100	0	23386	0	100	0	200	26559	0	100	0	200
	50	17417	0	100	0	150	19184	0	100	0	150
	100	11588	0	100	0	100	12647	0	100	0	100
1000	0	230424	0	1000	0	2000	242975	0	1000	0	2000
	50	173469	0	1000	0	1500	181545	0	1000	0	1500
	100	118679	0	1000	0	1000	121095	0	1000	0	1000
		run 5									
sample size % correct		speed (ms)	failed	validated	corrected	#requests					
10	0	2369	0	10	0	20					
	50	1852	0	10	0	15					
	100	1250	0	10	0	10					
100	0	20614	0	100	0	200					
	50	15266	0	100	0	150					
	100	10140	0	100	0	100					
1000	0	242449	0	1000	0	2000					
	50	182099	0	1000	0	1500					
	100	119760	0	1000	0	1000					

Figure 19 - Test results location speed test (FavorNames False, Laptop)

E.4 PC FavorNames True test results

Favor names aan

		run 1					run 2				
sample size	% correct	speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	1090	0	0	10	10	419	0	0	10	10
	50	1071	0	5	5	10	426	0	5	5	10
	100	1089	0	10	0	10	413	0	10	0	10
100	0	2939	0	0	100	100	3272	0	0	100	100
	50	2932	0	50	50	100	3043	0	50	50	100
	100	3467	0	100	0	100	2977	0	100	0	100
1000	0	28158	0	0	1000	1000	35908	0	0	1000	1000
	50	29038	0	500	500	1000	28837	0	500	500	1000
	100	28376	0	1000	0	1000	28809	0	1000	0	1000
		run 3					run 4				
sample size	% correct	speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	460	0	0	10	10	869	0	0	10	10
	50	439	0	5	5	10	864	0	5	5	10
	100	419	0	10	0	10	857	0	10	0	10
100	0	3000	0	0	100	100	2925	0	0	100	100
	50	3115	0	50	50	100	3038	0	50	50	100
	100	2878	0	100	0	100	2871	0	100	0	100
1000	0	36203	0	0	1000	1000	28037	0	0	1000	1000
	50	28045	0	50	50	1000	28091	0	500	500	1000
	100	42365	0	1000	0	1000	27588	0	1000	0	1000
		run 5									
sample size	% correct	speed (ms)	failed	validated	corrected	#requests					
10	0	439	0	0	10	10					
	50	410	0	5	5	10					
	100	410	0	10	0	10					
100	0	2783	0	0	100	100					
	50	2794	0	50	50	100					
	100	2969	0	100	0	100					
1000	0	32045	0	0	1000	1000					
	50	37625	0	500	500	1000					
	100	29155	0	1000	0	1000					

Figure 20 - Test results location speed test (FavorNames True, PC)

E.5 PC FavorNames False test results

Favor names uit

		run 1					run 2				
sample size % correct		speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	1391	0	10	0	20	692	0	10	0	20
	50	1209	0	10	0	15	523	0	10	0	15
	100	1062	0	10	0	10	421	0	10	0	10
100	0	5449	0	100	0	200	5688	0	100	0	200
	50	4210	0	100	0	150	4140	0	100	0	150
	100	2848	0	100	0	100	2842	0	100	0	100
1000	0	63787	0	1000	0	2000	61446	0	1000	0	2000
	50	45405	0	1000	0	1500	43576	0	1000	0	1500
	100	29534	0	1000	0	1000	28070	0	1000	0	1000
		run 3					run 4				
sample size % correct		speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	703	0	10	0	20	1065	0	10	0	20
	50	564	0	10	0	15	918	0	10	0	15
	100	380	0	10	0	10	751	0	10	0	10
100	0	5618	0	100	0	200	5583	0	100	0	200
	50	4108	0	100	0	150	4323	1	99	0	150
	100	2836	0	100	0	100	3047	0	100	0	100
1000	0	55872	0	1000	0	2000	57428	0	1000	0	2000
	50	44025	0	1000	0	1500	44637	0	1000	0	1500
	100	31341	0	1000	0	1000	30902	0	1000	0	1000
		run 5									
sample size % correct		speed (ms)	failed	validated	corrected	#requests					
10	0	675	0	10	0	20					
	50	522	0	10	0	15					
	100	401	0	10	0	10					
100	0	6129	0	100	0	200					
	50	4851	0	100	0	150					
	100	2818	0	100	0	100					
1000	0	56584	0	1000	0	2000					
	50	50697	0	1000	0	1500					
	100	26823	0	1000	0	1000					

Figure 21 - Test results location speed test (FavorNames False, PC)

E.6 Server FavorNames True test results

Favor names aan

		run 1					run 2				
sample size	% correct	speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	374	0	0	10	10	401	0	0	10	10
	50	372	0	5	5	10	426	0	5	5	10
	100	378	0	10	0	10	404	0	10	0	10
100	0	3046	0	0	100	100	3691	0	0	100	100
	50	3092	0	50	50	100	3656	0	50	50	100
	100	3079	0	100	0	100	3637	0	100	0	100
1000	0	31786	0	0	1000	1000	33385	0	0	1000	1000
	50	28846	0	50	50	1000	38383	0	50	50	1000
	100	38241	0	1000	0	1000	28997	0	1000	0	1000
		run 3					run 4				
sample size	% correct	speed (ms)	failed	validated	corrected	#requests	speed (ms)	failed	validated	corrected	#requests
10	0	416	0	0	10	10	383	0	0	10	10
	50	408	0	5	5	10	383	0	5	5	10
	100	417	0	10	0	10	386	0	10	0	10
100	0	3433	0	0	100	100	3382	0	0	100	100
	50	3665	0	50	50	100	3311	0	50	50	100
	100	3636	0	100	0	100	3347	0	100	0	100
1000	0	32355	0	0	1000	1000	31889	0	0	1000	1000
	50	32465	0	50	50	1000	32320	0	50	50	1000
	100	32223	0	1000	0	1000	33177	0	1000	0	1000
		run 5									
sample size	% correct	speed (ms)	failed	validated	corrected	#requests					
10	0	386	0	0	10	10					
	50	381	0	5	5	10					
	100	395	0	10	0	10					
100	0	3159	0	0	100	100					
	50	3277	0	50	50	100					
	100	3216	0	100	0	100					
1000	0	29926	0	0	1000	1000					
	50	29132	0	50	50	1000					
	100	29560	0	1000	0	1000					

Figure 22 - Test results location speed test (FavorNames True, Server)

Appendix F - Database module tests

F.1 Specifications of the systems

LAPTOP	specs	Processor name	Intel Core i7 7700HQ
		CPU max speed	2.80 GHz
		CPU L3 cache size	6 MB
		CPU number of cores	4
		CPU number of threads	8
		Ram type	DDR4
		RAM size	2 x 8 GB
		RAM speed	2400 MHz
		Storage type	HDD
		Network speed	44,17 Mbps/ 19,91 Mbps (Download / Upload)
		Network ping	12 ms
		Network type	WIFI (2.4 GHz)
PC	specs	Processor name	Intel Core i7 4770K
		CPU max speed	3.90 GHz
		CPU L3 cache size	8 MB
		CPU number of cores	4
		CPU number of threads	8
		Ram type	DDR3
		RAM size	2 x 8 GB
		RAM speed	1333 MHz
		Storage type	SSD
		Network speed	92,89 Mbps/ 87,18 Mbps (Download / Upload)
		Network ping	9 ms
		Network type	LAN
Server	specs	Processor name	Intel Xeon E3-1240L v5
		CPU max speed	3.20 GHz
		CPU L3 cache size	8 MB
		CPU number of cores	4
		CPU number of threads	8
		Ram type	DDR4
		RAM size	2 x 8 GB
		RAM speed	2133MHz
		Storage type	SSD
		Network speed	100,00 Mbps/ 100,00 Mbps (Download / Upload)
		Network ping	3 ms
		Network type	LAN (local)

Table 16 - System specifications

F.2 Database module speed test Laptop

		run 1			run 2		
sample size	doubles %	speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	27953	1000	1000	24721	1000	1000
	50	17941	1000	500	14661	1000	500
	100	17972	1000	2	14715	1000	2
10000	0	257227	10000	10000	235138	10000	10000
	50	197907	10000	5000	165587	10000	5000
	100	158240	10000	2	133501	10000	2
100000	0	3914936	100000	100000	3641055	100000	100000
	50	3342622	100000	50000	3115555	100000	50000
	100	1616566	100000	2	1266266	100000	2
		run 3			run 4		
sample size	doubles %	speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	24538	1000	1000	24803	1000	1000
	50	14612	1000	500	14720	1000	500
	100	13577	1000	2	13600	1000	2
10000	0	230560	10000	10000	230546	10000	10000
	50	171031	10000	5000	165506	10000	5000
	100	134542	10000	2	128986	10000	2
100000	0	3609658	100000	100000	3609993	100000	100000
	50	3082576	100000	50000	3094976	100000	50000
	100	1275169	100000	2	1332797	100000	2
		run 5			run 6		
sample size	doubles %	speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	24701	1000	1000	24598	1000	1000
	50	14734	1000	500	14633	1000	500
	100	13567	1000	2	13570	1000	2
10000	0	234102	10000	10000	231761	10000	10000
	50	168763	10000	5000	163337	10000	5000
	100	127917	10000	2	135664	10000	2
100000	0	3645310	100000	100000	3708186	100000	100000
	50	3110174	100000	50000	3182325	100000	50000
	100	1333830	100000	2	1362636	100000	2

Figure 24 - Test results Database speed test (Laptop)

F.3 Database module speed test PC

		run 1			run 2		
sample size doubles %		speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	20127	1000	1000	21130	1000	1000
	50	13543	1000	500	11249	1000	500
	100	9108	1000	2	7977	1000	2
10000	0	182504	10000	10000	178006	10000	10000
	50	98884	10000	5000	102115	10000	5000
	100	71432	10000	2	72285	10000	2
100000	0	3107980	100000	100000	2965743	100000	100000
	50	2415035	100000	50000	2427787	100000	50000
	100	708798	100000	2	709681	100000	2
		run 3			run 4		
sample size doubles %		speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	20159	1000	1000	20161	1000	1000
	50	14646	1000	500	11247	1000	500
	100	10256	1000	2	10292	1000	2
10000	0	186977	10000	10000	185814	10000	10000
	50	103301	10000	5303	109871	10000	5000
	100	84507	10000	2	74765	10000	2
100000	0	2947667	100000	100000	2956386	100000	100000
	50	2544934	100000	50000	2389985	100000	50000
	100	774051	100000	2	709085	100000	2
		run 5			run 6		
sample size doubles %		speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	20078	1000	1000	20069	1000	1000
	50	14620	1000	500	13465	1000	500
	100	9058	1000	2	8009	1000	2
10000	0	187844	10000	10000	181468	10000	10000
	50	106397	10000	5000	99886	10000	5000
	100	73332	10000	2	71395	10000	2
100000	0	2902790	100000	100000	2985435	100000	100000
	50	2381359	100000	50000	2452993	100000	50000
	100	800882	100000	2	716182	100000	2

Figure 25 - Test results database speed test (PC)

F.4 Database module speed test Server

		run 1			run 2		
sample size	doubles %	speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	12394	1000	1000	12329	1000	1000
	50	12322	1000	976	12331	1000	979
	100	1297	1000	34	1320	1000	27
10000	0	111931	10000	10000	111931	10000	10000
	50	109693	10000	9820	109693	10000	9820
	100	4118	10000	2	4137	10000	2
100000	0	1132985	100000	100000	1131848	100000	100000
	50	1086829	100000	94721	1087978	100000	94974
	100	16398	100000	2	16396	100000	2
		run 3			run 4		
sample size	doubles %	speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	12295	1000	1000	12389	1000	1000
	50	12295	1000	977	12335	1000	971
	100	1332	1000	35	1335	1000	37
10000	0	110913	10000	10000	111890	10000	10000
	50	108624	10000	9821	109722	10000	9833
	100	4314	10000	2	4080	10000	2
100000	0	1130730	100000	100000	1138609	100000	100000
	50	1082377	100000	94481	1094552	100000	95077
	100	16519	100000	2	16491	100000	2
		run 5			run 6		
sample size	doubles %	speed (ms)	# selects	# inserts	speed (ms)	# selects	# inserts
1000	0	12323	1000	1000	12307	1000	1000
	50	12310	1000	982	12315	1000	978
	100	1341	1000	16	1337	1000	12
10000	0	109732	10000	10000	114189	10000	10000
	50	108724	10000	9835	111948	10000	9824
	100	4200	10000	2	4093	10000	2
100000	0	1135183	100000	100000	1136330	100000	100000
	50	1090066	100000	94840	1084558	100000	94140
	100	16479	100000	2	17581	100000	3

Figure 26 - Test results database speed test (Server)